

# **Attunity Connect Developer SDK**

Version 4.1



## **Attunity Connect Developer SDK**

© 2003 by Attunity Ltd.

Due to a policy of continuous development, Attunity Ltd. reserves the right to alter, without prior notice, the specifications and descriptions outlined in this document. No part of this document shall be deemed to be part of any contract or warranty whatsoever.

Attunity Ltd. retains the sole proprietary rights to all information contained in this document. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopy, recording, or otherwise, without prior written permission of Attunity Ltd. or its duly appointed authorized representatives.

Product names mentioned in this document are for identification purposes only and may be trademarks or registered trademarks of their respective companies.

# Table of Contents

<b>About this Manual</b>	<b>13</b>
<b>Part I Starting to Use the Developer SDK</b>	
<b>Chapter 1: Setting Up the Developer SDK</b>	<b>19</b>
The Attunity Connect Developer SDK Contents.....	19
Enabling the Developer SDK .....	20
Sample addon.def File for a Data Driver .....	20
Sample addon.def File for an Application Adapter .....	21
Defining Multiple “Add-Ons”.....	21
<b>Chapter 2: Creating Data Drivers and Application Adapters</b>	<b>23</b>
Defining a Data Driver .....	23
Defining an Application Adapter .....	24
Registering Drivers and Adapters to Attunity Connect .....	25
Setting Up Access to a Data Driver or Application Adapter .....	26
Tracing the Flow of Operations in a Data Driver .....	28
<b>Part II Data Drivers</b>	
<b>Chapter 3: The Data Driver (GDB) API</b>	<b>33</b>
Loading a Driver .....	34
The LOAD Function.....	34
The UNLOAD Function.....	35
Connecting to a Data Source .....	35
The CONNECT Function .....	35
The SET_CONNECT Function .....	36
The DISCONNECT Function.....	38
Defining Metadata .....	38
The GET_ITEM_LIST Function .....	39
The DESCRIBE_TABLE Function .....	40
The DESCRIBE_SP Function .....	41
The GET_EXTENDED_METADATA Function .....	42
The GET_FILE_INFO Function .....	43

Retrieving Data .....	44
The OPEN Function.....	44
The SET_BUFFER Function.....	45
The SET_INDEX Function.....	46
The SET_FILTER Function .....	48
The SET_FILTER_PARAMS Function.....	50
The SET_PARAMS Function .....	51
The SET_BM Function .....	51
The CONVERT_BM_DATA Function.....	53
The FETCH Function .....	53
The CLOSE Function.....	54
The REWIND Function .....	55
Updating Data.....	56
The ADD_ROW Function .....	56
The UPDATE_ROW Function.....	56
The DELETE_ROW Function .....	57
The LOCK_ROW Function.....	58
The UNLOCK_ROW Function.....	58
Managing Transactions .....	59
The START_TRANSACTION Function.....	59
The COMMIT_TRANSACTION Function.....	60
The ROLLBACK_TRANSACTION Function .....	60
The START_DISTRIBUTED_TRANSACTION Function.....	61
The PREPARE_COMMIT_TRANSACTION Function .....	61
The RECOVER_TRANSACTION Function.....	61
Data Definition.....	61
The CREATE_TABLE Function.....	62
The CREATE_INDEX Function.....	63
The DROP_TABLE Function .....	64
The DROP_INDEX Function.....	65
BLOB Support .....	65
The BLOB_OPEN Function .....	65
The BLOB_READ Function.....	66
The BLOB_WRITE Function.....	67
The BLOB_SEEK Function .....	68
The BLOB_CLOSE Function .....	68
SQL and non-SQL Command Support.....	68
The PREPARE_COMMAND Function .....	69
The DESCRIBE_COMMAND Function .....	69
The EXECUTE_IMMEDIATE Function.....	70
The EXECUTE_WITH_PARAMS Function .....	71

---

The FREE_COMMAND Function.....	72
The GET_DATE_LITERAL Function.....	72
Chapter Support.....	73
The DESCRIBE CHAPTER Function.....	73
The OPEN CHAPTER Function.....	74
The SET_ACTIVE CHAPTER Function .....	75
Error Functions .....	75
The GET_LAST_ERROR Function.....	76
<b>Chapter 4: Security Hooks</b>	<b>77</b>
<b>Chapter 5: Defining a Custom ODBC Driver</b>	<b>83</b>
Specifying ODBC Functions .....	83
Registering a Custom ODBC Driver to Attunity Connect.....	83
ODBC Functions .....	85
SQLAllocConnect .....	85
SQLAllocEnv .....	85
SQLAllocStmt.....	85
SQLBindCol.....	85
SQLBindParameter .....	85
SQLColumns .....	86
SQLConnect .....	86
SQLDescribeCol .....	86
SQLDescribeParam .....	86
SQLDisconnect.....	87
SQLDriverConnect.....	87
SQLError .....	87
SQLExecDirect .....	87
SQLExecute.....	87
SQLExtendedFetch.....	87
SQLFetch.....	88
SQLForeignKeys .....	88
SQLFreeConnect .....	88
SQLFreeEnv.....	88
SQLFreeStmt .....	88
SQLGetConnectOption .....	88
SQLGetData.....	89
SQLGetFunctions .....	89
SQLGetInfo .....	89
SQLGetTypeInfo .....	89
SQLNumParams.....	89

SQLNumResultCols .....	89
SQLParamData .....	90
SQLPrepare .....	90
SQLPrimaryKeys .....	90
SQLProcedureColumns .....	90
SQLProcedures .....	90
SQLPutData .....	91
SQLRowCount .....	91
SQLSetConnectOption .....	91
SQLSetStmtOption .....	91
SQLStatistics .....	91
SQLTables .....	92
SQLTransact .....	92

## Part III Application Adapters

<b>Chapter 6: The Application Adapter XML Definition Document</b>	<b>95</b>
The adapter Element .....	98
The interaction Element .....	100
The schema Element .....	100
The enumeration Element .....	101
The record Element .....	101
The variant record Element .....	103
The field Element .....	103
The Type Schema .....	105
<b>Chapter 7: The Application Adapter API (GAP)</b>	<b>109</b>
The INITIALIZE Function .....	112
Connection APIs .....	113
The CONNECT Function .....	113
The SET_CONNECTION Function .....	113
The DISCONNECT Function .....	114
The REAUTHENTICATE Function .....	114
The CLEAN_CONNECTION Function .....	115
The HOLD_CONNECTION Function .....	115
Transaction APIs .....	116
The SET_AUTO_COMMIT Function .....	116
The TRANSACTION_START Function .....	116
The TRANSACTION_END Function .....	117
The TRANSACTION_COMMIT Function .....	117
The TRANSACTION_PREPARE Function .....	118

---

The TRANSACTION_ROLLBACK Function .....	118
The TRANSACTION_FORGET Function .....	118
The TRANSACTION_RECOVER Function .....	119
The EXECUTE Function .....	119
Metadata APIs .....	120
The GET_METADATA_ITEM Function.....	120
The GET_METADATA_LIST Function.....	121
The PING Function.....	121
The BIND Function.....	121
Exception Handling .....	122
The GAP_SET_EXCEPTION[_V] Functions.....	123
The Adapter Event and Handler Functions .....	124
The GAP_SET_INTERACTION_INFO Function .....	124
The GAP_GET_INTERACTION_INFO Function.....	125
The GAP_GET_EXECUTE_INFO Function .....	125
The GAP_SET_ADAPTER_INFO Function .....	126
The GAP_GET_ADAPTER_INFO Function.....	126
The GAP_GET_METADATA_ATTR Function.....	126

## Part IV User Defined Data Types

<b>Chapter 8: Defining Data Types</b>	<b>131</b>
Defining a Data Type .....	131
Registering a User-Defined Data Type to Attunity Connect.....	132
Data Type Functions.....	133
Conversion Functions .....	133
The TO_STRING Function.....	134
The FROM_STRING Function.....	134
The TO_BASE Function .....	135
The FROM_BASE Function .....	135
The CONVERT_TO Function.....	136
The TO_EXPOSED Function .....	137
The WIDTH_TO_SIZE Function.....	139
Comparison Functions .....	140
The COMPARE Function .....	140
The IS_EMPTY_VAL Function .....	141
The IS_NULL_VAL Function.....	142

Initialization/Assignment Functions.....	142
The SET_EMPTY_VAL Function.....	143
The SET_NULL_VAL Function .....	143
The LOWEST_VAL Function.....	144
The HIGHEST_VAL Function .....	144
The RANDOM_VAL Function.....	145
Error Functions .....	146
The GET_LAST_ERROR Function .....	146
Data Type Support Macros .....	147

## Part V Additional SDK Features

<b>Chapter 9: Registering a Startup Function</b>	<b>151</b>
<b>Chapter 10: Developer SDK Predefined Functions</b>	<b>153</b>
Data Type Support Functions .....	154
DT_CONVERT .....	154
DT_ADD_DATATYPE .....	154
Memory Pool Functions .....	155
MEMP_NEW .....	155
MEMP_MALLOC .....	156
MEMP_MALLOC8 .....	156
MEMP_FREE .....	156
MEMP_FREE_POOL.....	157
MEMP_DELETE.....	157
Dynamic Array Functions.....	157
ARR_NEW .....	158
ARR_RESET.....	158
ARR_SIZE.....	159
ARR_DELETE.....	159
ARR_INSERT.....	159
ARR_ADD.....	160
ARR_SET_ENTRY.....	161
ARR_COPY.....	161
ARR_FREE.....	162
COMPARE.....	162
ARR_BINSEARCH .....	163
ARR_BSEARCH.....	164

---

Dynamic String Functions.....	164
DSTR_NEW.....	165
DSTR_APPEND.....	165
DSTR_M_APPEND.....	166
DSTR_RESET .....	166
DSTR_LENGTH.....	166
DSTR_INSERT .....	167
DSTR_CUT.....	167
DSTR_GET_ITEM .....	168
General String Functions .....	168
STR_C_STRING.....	168
STR_C_STRING_PAD.....	169
STR_STRCONCAT .....	169
Filter Functions.....	170
TRAVERSE_FILTER.....	170
Metadata Support Functions.....	170
GDBH_DESCRIBE_TABLE.....	171
GDBH_DESCRIBE_SP.....	171
GDBH_GET_ITEM_LIST.....	172
GDB_GET_TABLE_DA .....	173
NLS Functions .....	174
NLS_CVTFNC.....	174
RESET .....	174
MBLEN.....	174
CP_INFO .....	175
CP_STRCHR .....	175
CP_STRSTR .....	175
CP_STRICTCMP.....	175
CP_UPPERCASE.....	175
CP_STRNWCMP.....	176
CP_STRNICMP.....	176
CP_TOUPPER.....	176
CP_LOWERCASE.....	176
Code Page Support Functions .....	176
CP_REGISTER_CVT .....	177
CP_GET_ID .....	177
CP_REGISTER .....	177
Profile Functions .....	177
GDB_PROF_INIT_FILE.....	178
GDB_PROF_SECTION_FIRST.....	178
GDB_PROF_SECTION_NEXT .....	179
GDB_PROF_ITEM_FIRST.....	179

GDB_PROF_ITEM_NEXT.....	180
GDB_PROF_GET_VALUE.....	180
GDB_PROF_GET_CUR_VALUE.....	181
GDB_PROF_SET_SECTION.....	181
GDB_PROF_SET_ITEM.....	181
GDB_PROF_DELETE .....	182
Database Property Functions.....	182
DB_GET_PROPERTY.....	183
DB_GET_PROPERTY_STRING .....	183
DB_GET_PROPERTY_LONG.....	184
DB_SET_PROPERTY .....	184
<b>Chapter 11: National Language Support</b>	<b>187</b>

## Part A Appendix – Data Driver Example

<b>Appendix A: Sample Data Drivers</b>	<b>193</b>
Skeleton Data Driver .....	193
Setup .....	193
Connect .....	194
Disconnect.....	195
Open.....	195
Set Buffer.....	196
Set Index.....	196
Fetch .....	197
Close.....	197
Rewind .....	198
Get Last Error .....	198
Load .....	198
The Driver Skeleton Metadata (.adl) .....	200
The Driver Skeleton ADDON.DEF Entry.....	200
The Driver Skeleton Binding File Entry.....	200
DISAM-Like Data Driver .....	201

## Part B Appendix – Application Adapter Examples

<b>Appendix B: The Calc Sample Application Adapter</b>	<b>229</b>
The Application Adapter Definition.....	229
The ADDON.DEF File.....	229
Binding Definition .....	229
ACX Schema for the Calc Adapter .....	229
The C Code for the Calc Adapter .....	231
Example Input to the Calc Demo Adapter.....	234
Example Output from the Calc Demo Adapter .....	235
<b>Appendix C: The Mailer Sample Adapter</b>	<b>237</b>
The ADDON.DEF File.....	237
Binding Definition .....	237
The C++ Code for the Mailer Adapter .....	237
ACX Schema for Mailer Adapter.....	242
Example Input to the Mailer Demo Adapter.....	243
Example Output from the Mailer Demo Adapter .....	244

## Part C Appendix – User Defined Data Type Example

<b>Appendix D: Sample RVMS Data Type</b>	<b>247</b>
The RVMS Data Type Source Code .....	247
The Data Type ADDON.DEF Entry .....	250

## Part D Appendix – System Notes and SDK Data Structures

<b>Appendix E: System Notes</b>	<b>253</b>
<b>Appendix F: Data Structures</b>	<b>255</b>
Function Structures .....	255
GDB_DB_FUNCTIONS.....	255
GDB_HELP_FUNCTIONS.....	263
ODBC_FUNCTIONS .....	265
Data Buffer Structures .....	266
GDB_VAL_DESC.....	266
GDB_ITEM_DESC.....	267

GDB_FILE_INFO .....	268
GDB_BUFFER .....	270
GDB_BUFFER_COLUMN .....	270
Metadata Structures .....	271
TABLE_DA .....	272
COLUMN_DA .....	275
INDEX_DA .....	280
SEG_DA .....	282
Attunity Connect Internal Structures .....	283
Filter Structure .....	284
GDB_FTREE_NODE .....	284
Data Type Structure .....	290
UDT_DATATYPE .....	290
Attributes .....	294
Driver Attributes (GDB_DB_FUNCTIONS) .....	294
Table Attributes (TABLE_DA) .....	301
Column Attributes (COLUMN_DA) .....	301
Index Attributes (INDEX_DA) .....	302
Index Segment Attributes (SEG_DA) .....	303
Data Type Attributes (UDT_DATATYPE) .....	303
Return Status Codes .....	305
GDB_STATUS .....	305
UDT_STATUS .....	306

# About this Manual

Attunity Connect is an information infrastructure solution that provides built-in connectivity to data sources and applications across a distributed environment that can encompass different platforms, networks, and even the Internet.

In addition to a range of ready-to-implement adapters and integration software, Attunity Connect includes development and management tools that make it easy to add applications and data sources to the integrated enterprise.

Attunity Connect provides a flexible, dependable infrastructure for web-enabling mainframe and other legacy solutions. The architecture takes advantage of distributed processing and an array of industry standards (such as XML, JCA, JDBC, ODBC and OLE DB).

This manual describes the Attunity Connect SDK, used to build data drivers and application adapters and consists of the following parts:

## **Part I, Starting to Use the Developer SDK**

This part describes the how to set up the SDK and procedures for creating a data driver or application adapter, using the SDK.

## **Part II, Data Drivers**

This part describes the APIs that are used to build a data driver.

## **Part III, Application Adapters**

This part describes the APIs that are used to build an application adapter.

## **Part IV, User Defined Data Types**

This part describes how to define a customized data type for use in either a data or application adapter.

## **Part V, Additional SDK Features**

This part describes SDK features such as start-up functions and pre-defined functions that can be incorporated in a custom-built data or application adapter.

## Appendices

The appendices provide examples of the SDK.

### Intended Audience

This manual is intended for developers who need to build a driver to data or adapter to an application.

### Typographic Conventions

This document uses the following typographic conventions:

***Italics*** Italics denotes either a placeholder/variable for an actual value or a new term.

**►To do** A bolded sentence starting with the word To is followed by a procedure.

❖ This symbol prefaces a note to the main text.

**Sample code** This font denotes sample code and commands that you type on the keyboard.

**Xyz Platforms** Grey shading highlights platform-specific information.

### Conventions for Commands

The following conventions are used to describe commands entered to the keyboard or console and the syntax used to define Attunity Connect metadata using the Attunity Connect Data Dictionary (ADD):

- Keywords are shown in standard type. Variables are shown as italics.
- Square brackets ([] ) enclose optional items.
- Vertical lines (|) separate between choices.
- Ellipses (...) indicate that the preceding item can be repeated.

### Related Documentation

The Attunity Connect documentation set includes the following:

#### *Attunity Connect Release Notes*

New features for the current Attunity Connect version.

#### *Getting Started with Attunity Connect*

A walk-through describing how to connect to data using Attunity Connect.

#### *Attunity Connect Installation Guides*

Guides describing how to install Attunity Connect on Compaq NonStop, OpenVMS, OS/390 and z/OS, OS/400, UNIX and Windows platforms.

#### *Attunity Connect Reference*

A complete reference to Attunity Connect.

#### *Attunity Connect Guide*

A guide to using Attunity Connect application adapters and data sources on all platforms.

#### *Attunity Connect and Applications*

Tutorials demonstrating how to use Attunity Connect with various applications, such as IBM WebSphere.

*Using the Attunity Connect Syntax File  
(NAV.SYN)*

Documentation detailing how to use the Attunity Connect syntax file to control the way SQL is processed by Attunity Connect.

This manual.

*Attunity Connect Developer SDK*



# **Part I**

## **Starting to Use the Developer SDK**



# Chapter 1 Setting Up the Developer SDK

This chapter describes the following:

- The Attunity Connect Developer SDK Contents
- Enabling the Developer SDK

## The Attunity Connect Developer SDK Contents

The Attunity Connect Developer SDK contains the components you use to develop custom adapters to data and applications. The SDK contains different groups of header files appropriate to what you are developing.

These files are located in the INCLUDE directory under the directory where Attunity Connect is installed (on Compaq NonStop Himalaya platforms, in the subvolume where Attunity Connect is installed and under OS/390 the file is identified by the high level qualifier where Attunity Connect is installed).

**The SDK Contents for Data Drivers** Attunity Connect supplies the following header file for the Developer SDK for data drivers:

- **dbgdb.h** – This file defines the functions, structures and macros for the GDB API (to add new drivers).

**The SDK Contents for Application Adapters** Attunity Connect supplies the following header files for the Developer SDK for application adapters:

- **xmlproto.h** – A collection of services that facilitate the interaction between XML-based protocols and C programming.
- **gap.h** – A file defining the interfaces for adding new application adapter types to Attunity Connect.

**The SDK Contents for user defined data types and ODBC data source administration** Attunity Connect supplies the following header files for the Developer SDK for user defined data types and ODBC data source administration:

- **dtypeid.h** – This file lists the data type codes for predefined data types in Attunity Connect. You use information from this file for a custom data type or for a custom driver that supplies its own metadata.
- **odbcfn.h** – This file defines the functions, structures and macros for the generic ODBC API. This interface allows you to define a custom ODBC driver to an ODBC backend data source on a

non-Windows platform (when ODBC data source administration is not available).

## Enabling the Developer SDK

When Attunity Connect initializes, it reads and processes the definitions in a file called *addon.def*. The *addon.def* file contains the definitions for data drivers, application adapters, user-defined data types, startup functions, and custom ODBC drivers. Any development done using the SDK is defined to the Attunity Connect environment in *addon.def*.

- ❖ The *addon.def* file is located in the DEF directory under the directory where Attunity Connect is installed. (On Tandem platforms the file is called *addondef* and is located in the subvolume where Attunity Connect is installed. Under OS/390 the file – *addondef* – is identified by the high level qualifier where Attunity Connect is installed.)

The basic syntax for a definition in the *addon.def* file is:

```
[name]
TYPE=type
item = value
...
```

where:

**name** – A unique identifier for the Data Driver, Application Adapter, startup function, ODBC driver, or data type.

**type** – DRIVER, DATATYPE, STARTUP, ODBC or ADAPTER.

**item / value** – A series of keyword and value assignments appropriate to the type of SDK “add-on” being defined:

- DRIVER: For details, see page 25.
- DATATYPE: For details, see page 131.
- STARTUP: For details, see page 151.
- ODBC: For details, see page 83.
- ADAPTER: For details, see page 25.

## Sample *addon.def* File for a Data Driver

The following is an example of an *addon.def* file that defines a new data driver (called DB\_FILE) and a startup function to register new data types:

```
[DB_FILE]
TYPE=DRIVER
```

```
INIT-FUNCTION=db_file_get_functions
SHAREABLE-NAME=NVDB_FILE

[MYDTS]
TYPE=STARTUP
INIT-FUNCTION=add_rvms_date_dt
SHAREABLE-NAME=MY_DATATYPES
```

## Sample addon.def File for an Application Adapter

The following is an example of an addon.def file that defines a new application adapter (called MYAPP):

```
[myapp]
type=adapter
INIT-FUNCTION=register_myAPP_adapter
SHAREABLE-NAME=d:\myapp\myapp.dll
```

## Defining Multiple “Add-Ons”

Attunity recommends that you prepare a separate define file for each new SDK “add-on” that you create, to avoid overwriting existing definitions in addon.def. The define file can contain one or more definition sections, with each section using the item/value syntax appropriate to the type of add-on being defined.

The Attunity Connect utility (NAV\_UTIL) includes an option, ADDON, which you can use to merge a new define file with addon.def. The ADDON utility adds or replaces sections by name, regardless of their content.

- ❖ The ADDON utility creates the addon.def file if it does not already exist.

Use the following command to merge “add-on” definitions:

```
nav_util addon define_file
```

where:

**define\_file** – An input text file containing one or more definition sections. Unless you specify a full path, NAV\_UTIL searches for the file in the current working directory.



# Chapter 2 Creating Data Drivers and Application Adapters

This chapter describes the following topics:

- Defining a Data Driver
- Defining an Application Adapter
- Registering Drivers and Adapters to Attunity Connect
- Setting Up Access to a Data Driver or Application Adapter
- Tracing the Flow of Operations in a Data Driver

Attunity supplies examples of data drivers and applications adapters and user defined data types in the appendices.

## Defining a Data Driver

This section outlines the steps that you must perform to define a data driver.

1. **Define metadata for the data source**
  - If using Attunity Connect metadata, create the ADD metadata in the Attunity Connect repository. For details, see the *Attunity Connect Reference*.
  - If not using Attunity Connect metadata, supply metadata routines as part of the driver. For details, see "Defining Metadata" on page 38.
2. **Write the data driver routines**

For details, see "The Data Driver (GDB) API" on page 33 and "Sample Data Drivers" on page 193.
3. **Write the external LOAD function**, to establish the function pointers and attributes for the data driver.

For details, see "The LOAD Function" on page 34 and "Load" on page 198 (in the sample driver skeleton).
4. **Compile all the functions into a DLL**

Use the appropriate commands for the platform and operating system.
5. **Register the driver to Attunity Connect**

For details, see "Registering Drivers and Adapters to Attunity Connect" on page 25.

- ❖ In this document, "DLL" is a generic term to refer to a shareable image or shared library of functions, as appropriate to the specific platform where Attunity Connect runs.

## 6. Set up access to the driver

For details, see "Setting Up Access to a Data Driver or Application Adapter" on page 26.

# Defining an Application Adapter

This section outlines the steps that you must perform to define an application adapter.

### 1. Provide the definition of the application adapter

Using the Attunity ACX XML protocol, define the interactions, functionality, and input/output structures of the adapter in an XML file (*adapter.xml*). For details refer to "The Application Adapter XML Definition Document" on page 95.

### 2. Run the protogen command

Use the following command line to generate a C header (.h) file, which defines the equivalent C language structures for the input/output XML records:

```
nav_util protogen abc.xml output_location [-verbose]  
where:
```

**protogen** – The Attunity Connect ACX compiler.

**abc.xml** – The application adapter definition (defined in step 2, above).

**output\_location** – The directory where the output C header (.h) file resides. (The name of this header file is defined in the application definition .xml file.)

**-verbose** – Displays the offset and size of every field generated.

### 3. Write the routines for the application adapter

Write and compile the DLL for the application adapter, which involves including the following files:

- the C source code
- the define (.def) files
- the C header (.h) files generated by the protogen command

For details, see "The Application Adapter API (GAP)" on page 109.

Also see the sample application adapters described in the appendixes.

**4. Register the adapter to Attunity Connect**

For details, see "Registering Drivers and Adapters to Attunity Connect", below.

**5. Set up access to the adapter**

For details, see "Setting Up Access to a Data Driver or Application Adapter" on page 26.

## Registering Drivers and Adapters to Attunity Connect

To register a data driver or application adapter in the Attunity Connect environment, the addon.def file must include a section defining the driver or adapter.

- ❖ Attunity Connect registers a driver or adapter when the addon.def file is processed, but does not load the driver or adapter until a data source or application of that type is accessed.

Create a define file with the “add-on” definitions and use the ADDON option of the NAV\_UTIL utility to merge the define file with existing entries in addon.def. For details, see page 21.

The format for the “add-on” definition file is:

```
[name]
TYPE=DRIVER
SHAREABLE-NAME=image
INIT-FUNCTION=function
type=type
```

where:

**name** – A unique identifier for the driver or adapter. This value is used in the binding configuration (see "Setting Up Access to a Data Driver or Application Adapter" on page 26).

- ❖ The name must be unique both in the entries in the addon.def file and the name specified in the <datasource> section (for a data driver) or <adapter> section (for an application adapter) of the binding configuration.

**image** – The path of the shareable image file (a DLL on Windows platforms). When binding to the first data source or application using the adapter, Attunity Connect locates the shareable image using platform-specific conventions. If Attunity Connect cannot locate the shareable image (for example, a specified environment variable or filename does not exist), Attunity Connect writes an error message in

the Attunity Connect log file. See "System Notes" on page 253 for additional details about this entry.

- ❖ In this document, "DLL" is a generic term to refer to a shareable image or shared library of functions, as appropriate to the specific platform where Attunity Connect runs.

**function** – The name of the initialization function defined in the data driver. After locating the shareable image, Attunity Connect loads and searches the shareable image for the initialization function.

- ❖ See "The LOAD Function" on page 34 for details about the parameters and return values that are expected for a data driver.
- ❖ The shareable image must expose the initialization function in order for Attunity Connect to reference the adapter. The initialization function specified cannot be static. Different platforms provide different mechanisms for exposing a function in a shareable image; consult the system documentation for details and see "System Notes" on page 253.

**type** – The type of the "add-on". Valid values are: DRIVER, DATATYPE, STARTUP, ODBC or ADAPTER.

See "The Driver Skeleton ADDON.DEF Entry" on page 200 for an example using a data driver.

## Setting Up Access to a Data Driver or Application Adapter

Access to a driver or adapter is defined in an Attunity Connect binding configuration. To edit the binding information, use the NAV\_UTIL EDIT utility, as follows:

```
NAV_UTIL EDIT BINDING binding_name
```

### Data Drivers

Set up access to a data source that will use a data driver by specifying connection information in an Attunity Connect binding configuration:

```
<bindings>
  <binding name="name">
    <datasources>
      <datasource name="name"
                  type="data_driver_type"
                  connect="connect_information" />
    </datasources>
    <adapters/>
  </binding>
</bindings>
```

where:

**name** – The name used to identify the data source to Attunity Connect and which applications use to access the data.

**data\_adapter\_type** – The name that you specify for the data driver in the addon.def file. For details, see "Registering Drivers and Adapters to Attunity Connect" on page 25.

**connect\_information** – The specific information Attunity Connect needs to connect to this data source. The connection information is supplied as input to the CONNECT function in the GDB API. (For details, see page 35.)

See "The Driver Skeleton Binding File Entry" on page 200 for an example.

**Application Adapter** Set up access to an application adapter by specifying the adapter in the Attunity Connect binding. An adapter is specified for a particular binding entry, as follows:

```
<bindings>
  <binding name="name">
    ...
    <adapters>
      <adapter name="name" type="type"
        connect="connect_string"
        definition="definition_name">
        <config config_parameters/>
      </adapter>
    </adapters>
  </binding>
</bindings>
```

where:

**name** – The name used to identify the application adapter to Attunity Connect.

**adapter\_type** – The name that you specify for the application adapter in the addon.def file. For details, see "Registering Drivers and Adapters to Attunity Connect" on page 25.

**connect** – Adapter-specific connect string. This is available for older versions of Attunity Connect. From Attunity Connect version 3.3, it is recommended to use the <config> statement (see below).

**definition** – The name of the definition (interactions and schema) used to define the adapter. If the value here is the same as the adapter name, it can be omitted. Note that some adapters have an internal definition and a value here must be omitted.

An <adapter> statement can include specific adapter configuration attributes via a <config> statement (see below).

#### <config> Statement

A <config> statement specifies configuration properties of an Attunity Connect application adapter. The configuration information is specific to each adapter type.

```
<config attribute="value"
        attribute="value"
        ... />
```

where:

**attribute** – The name of the configuration property.

**value** – The value of the configuration property.

## Tracing the Flow of Operations in a Data Driver

- ❖ Currently this functionality is not available for application adapters.

When developing or enhancing a data driver, you may want to trace the flow of operations at the GDB API level. To do this, you need to specify the gdbTrace debugging parameter in the Attunity Connect environment settings, as follows:

```
<debug gdbTrace="true" />
```

This causes Attunity Connect to write trace statements to the standard Attunity Connect log file. Upon loading a data driver, Attunity Connect writes to the log information about the version of the driver, the attributes that are set, and the functions supported.

- ❖ The default log file (NAV.LOG, or NAVLOG on Tandem platforms) is located in the TMP directory under the directory where Attunity Connect is installed (on Tandem platforms, in the subvolume where Attunity Connect is installed). Use the logFile attribute in the Attunity Connect environment settings to specify a different file or location, as follows: <debug logFile="pathname" />.

Under OS/390, the trace statements are written to the job specified in the StartupScript parameter of the Workspace section of the daemon configuration.

Trace statements are printed after a given API operation is called. The trace statements appear in the form:

```
<driver_name> <status_return>:API_operation_details
```

where:

*driver\_name* is the name specified in `addon.def`.

*status\_return* is one of the return values for `GDB_STATUS`.

*API\_operation* is the name of the function that was called.

*details* varies depending on the function. For example, the CONNECT function includes details about the connect string, and the OPEN function includes details about the table and stream.

**Sample Trace Outputs** The following example shows the tracing statements generated upon loading a data driver named *abc*.

```
<abc> < OK>: GDB version 210
<abc> GDB_DB_FUNCTION INFO:
    Attributes: Column Lists; Transactions; Updates;
    Case Sensitive; Extended Names.
    Bookmark Size = 256
    Datasource Fns: io_get_last_error; io_connect;
    io_disconnect.
    Metadata Fns: io_get_item_list; io_describe_table.
    Data Retrieval Fns: io_open; io_set_buffer;
    io_set_index; io_set_bm;
    io_fetch; io_close; io_rewind; io_convert_bm_data.
    Update Fns: io_add_row; io_update_row;
    io_delete_row; io_lock_row; io_unlock_row.
    Transaction Fns: io_start_transaction;
    io_commit_transaction; io_rollback_transaction.
    Data Definition Fns: io_create_table.
    Blob Fns: <None>.
    SQL Fns: <None>.
    Chapter Fns: <None>.
```

The following example shows tracing statements for the request “`SELECT * FROM parts`”, where the parts table belongs to a DBMS driver.

Result (in log file):

```
<DBMS> <OK> OPEN parts(0)
<DBMS> <OK> REWIND parts(0)
<DBMS> <OK> SET_BUFFER parts(0)
<DBMS> <OK> REWIND parts(0)
<DBMS> <OK> FETCH parts(0)
```

[the above message is repeated for number of records in the part table]

```
<DBMS> <EOS> FETCH parts(0)
<DBMS> <OK> CLOSE parts(0)
```

# **Part II**

# **Data Drivers**



# Chapter 3 The Data Driver (GDB) API

A data driver definition includes routines from some or all of the following categories:

- Loading a Driver (page 34)
- Connecting to a Data Source (page 35)
- Defining Metadata (page 38)
- Retrieving Data (page 44)
- Updating Data (page 56)
- Managing Transactions (page 59)
- Data Definition (page 61)
- BLOB Support (page 65)
- SQL and non-SQL Command Support (page 68)
- Chapter Support (page 73)
- Error Functions (page 75)

## Required Functions

As shown in the "Sample Data Drivers" on page 193, to create a data driver you must supply at least the following elements of the GDB API:

- LOAD function
- CONNECT function
- Metadata definitions
- OPEN function
- SET\_BUFFER function
- FETCH function
- CLOSE function
- REWIND function

You may also need to supply some or all of the other driver functions, depending on the features that the driver supports. For example, if the driver supports update operations, you must write driver routines to add, delete, update, lock, and unlock records in the data source.

## Loading a Driver

This section describes the following functions:

- The LOAD Function
- The UNLOAD Function

### The LOAD Function

**Assignment** Specify the name of this function as the INIT-FUNCTION value for the driver definition in **addon.def**. (See "Registering Drivers and Adapters to Attunity Connect" on page 25.)

**Description** Attunity Connect calls this function at initialization to load the driver when binding to the first data source using this driver type.

**Status** Required.

**Comments** Unlike other functions in the GDB API, this function must be exported (exposed) by the DLL. Attunity Connect searches the DLL for the load function using the name given in the INIT-FUNCTION assignment of the **addon.def** file.

- ❖ Typically, the load function name is the only symbol that the driver needs to expose. Different platforms provide different mechanisms for exposing a function in a DLL; consult the system documentation for details and see "System Notes" on page 253.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDB_FNC_LOAD) (
    GDB_DB_FUNCTIONS *gdb_db_functions,
    GDB_HELP_FUNCTIONS *gdb_help);
```

Parameter	Usage	Description
<code>gdb_db_functions</code>	Output	A pointer to an allocated structure (GDB_DB_FUNCTIONS) that the driver populates with the list of functions implementing the GDB API for the driver, as well as assignments for various driver attributes.
<code>gdb_help</code>	Input	A pointer to a structure (GDB_HELP_FUNCTIONS) that defines various predefined support functions supplied by Attunity Connect for use with the Developer SDK.

## The UNLOAD Function

**Assignment** Specify the name of this function as the **io\_unload** member of the **GDB\_DB\_FUNCTIONS** structure. (For details, see page 259.)

**Description** Attunity Connect calls this function when terminating a server, to allow the driver to release any resources it may be holding.

**Status** Optional but recommended as a standard practice, to avoid errors due to conflicts with locked or held resources.

**Comments** This function is called after all data sources referenced for the driver have been disconnected.

The possible return values from this API are **GDB\_OK\_** and **GDB\_NOT\_OK\_**. (See "GDB\_STATUS" on page 305.)

**Prototype** `GDB_STATUS (*GDB_FNC_UNLOAD) () ;`

## Connecting to a Data Source

This section describes the following functions:

- The CONNECT Function
- The SET\_CONNECT Function
- The DISCONNECT Function

### The CONNECT Function

**Assignment** Specify the name of this function as the **io\_connect** member of the **GDB\_DB\_FUNCTIONS** structure. (For details, see page 259.)

**Description** Attunity Connect calls this function to connect to a specific data source (also called a TDP).

**Status** Required.

**Comments** This function may be called several times, to connect to different data sources that are defined for this driver in the binding settings.

The possible return values from this API are **GDB\_OK\_** and **GDB\_NOT\_OK\_**. (See "GDB\_STATUS" on page 305.) If the return from this function is **GDB\_NOT\_OK\_**, the DISCONNECT function is called.

**Prototype** `GDB_STATUS (*GDB_FNC_CONNECT) ( GDB_HANDLE *gdb_handle, char *ds_name,`

```
char *connect_string,  
char *username_password);
```

Parameter	Usage	Description
gdb_handle	Output	The address of a void pointer which is used as a handle to the connection. The driver typically returns in this parameter the address of an internal structure in which all the driver-specific connection details are kept.
ds_name	Input	The name of the data source to which you want the driver to connect, in the form: <i>dsname\$num</i> where <i>dsname</i> is a data source name specified in the datasource section of the binding settings, and <i>num</i> identifies the instance of the data source connection. (The same data source may be connected more than once.) For example, if the data source called MYDATA is connected twice, the ds_name parameter could be MYDATAS1 or MYDATAS2.
connect_string	Input	The specific information Attunity Connect needs to connect to this data source, that is, the connect parameter specified for this data source in the binding settings.
username_password	Input	The username and password specified for this data source in the client user profile. For more information, see “Attunity Connect Security” in the <i>Attunity Connect Reference</i> .

## The SET\_CONNECT Function

**Assignment** Specify the name of this function as the **io\_set\_connect** member of the **GDB\_DB\_FUNCTIONS** structure. (For details, see page 259.)

**Description** Attunity Connect calls this function to pass a structure with connection-specific attributes.

**Status** Optional

**Comments** SET\_CONNECT is required for retrieving the name of a table owner. Support for table owner recognition in a data driver involves the following:

- Specifying the owner of the database in the binding configuration, as follows:

```
<datasource name="ds_name" type="ds_type"
            owner="owner" />
```

where *ds\_type* is the name defined for the data driver in the ADDON.DEF file.

- Specifying the SET\_CONNECT function in the GDB\_DB\_FUNCTIONS structure in the GET\_FUNCTIONS initialization function (see page 198).
- Specifying the GDB\_SET\_SUPPORT\_OWNER(fnc) macro in the GET\_FUNCTIONS initialization function.
- Passing the database name as the first parameter to the DB\_GET\_PROPERTY\_STRING function, as in the following:

```
static GDB_STATUS db_xxx_set_connect(GDB_HANDLE gdb_handle,
                                     GDB_CONNECT_ATTRIBUTES * connect_attrib)
{
    XXX_CONNECT *xxx_connect = gdb_handle ;

    xxx_connect->database = connect_attrib->database ;
    xxx_connect->default_owner = DB_GET_PROPERTY_STRING
        (xxx_connect->database, "DEFAULT_OWNER") ;

    return GDB_OK_ ;
}
```

**Prototype** GDB\_STATUS (\*GDB\_FNC\_SET\_CONNECT) (
 GDB\_HANDLE gdb\_handle,
 GDB\_CONNECT\_ATTRIBUTES \* connect\_attrib);

Parameter	Usage	Description
gdb_handle	Input	Pointer to handle of the connection.
connect_attrib	Output	Pointer to internal structure (GDB_CONNECT_ATTRIBUTES), which includes an element specifying the database.

## The DISCONNECT Function

**Assignment** Specify the name of this function as the `io_disconnect` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 259.)

**Description** Attunity Connect calls this function to disconnect from a particular data source (TDP) to which the driver has previously connected using the CONNECT function.

**Status** Optional.

**Comments** The DISCONNECT function is called after all open streams are closed.

The driver should release all resources associated with the data source at this point. Some drivers, such as file system drivers, may not need to perform any tasks upon disconnect.

The possible return values from this API are `GDB_OK_` and `GDB_NOT_OK_`. (See "GDB\_STATUS" on page 305.)

**Prototype** `GDB_STATUS (*GDB_FNC_DISCONNECT) (`  
    `GDB_HANDLE gdb_handle);`

---

Parameter	Usage	Description
-----------	-------	-------------

<code>gdb_handle</code>	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
-------------------------	-------	---

## Defining Metadata

Attunity Connect requires metadata about the tables and procedures it can access. Attunity Connect metadata includes a list of tables and procedures, a description of the columns and data types that make up a table or procedure, as well as index information and statistics.

You can rely on the Attunity Connect proprietary Attunity Connect Data Dictionary (ADD) and its utilities to define and store metadata, or the data driver can provide this information directly, by implementing the API functions described in this section.

If you do not use Attunity Connect ADD for the driver's metadata, you must provide descriptions of the columns, indices, and tables to allow proper functioning within Attunity Connect. When such information is needed, Attunity Connect calls the following metadata-related driver functions that you supply:

- The GET\_ITEM\_LIST Function
- The DESCRIBE\_TABLE Function

- The DESCRIBE\_SP Function
- The GET\_EXTENDED\_METADATA Function
- The GET\_FILE\_INFO Function
- The DESCRIBE\_COMMAND Function
- The DESCRIBE\_CHAPTER Function

The GDB API includes definitions for several data structures that the driver must allocate when supplying metadata to Attunity Connect. These data structures are described in "Data Structures" on page 255.

## The GET\_ITEM\_LIST Function

**Assignment** Specify the name of this function as the `io_get_item_list` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 259.)

**Description** This function provides Attunity Connect with a list of selected items, such as tables, procedures, or views, in the data source, without getting all the metadata for each item.

**Status** Required for a non-ADD driver (where the driver supplies its own metadata to Attunity Connect).

Otherwise, the LOAD function should issue the macro `GDB_SET_ADD` and the driver should not implement this function.

**Comments** Attunity Connect first calls this function with a request to open a stream of item descriptions (`GDB_ITEM_LIST_OPEN_`). Then, Attunity Connect calls the function repeatedly with a series of GET requests to retrieve the list of requested items (`GDB_ITEM_LIST_GET_`), until `GDB_END_OF_STREAM_` is returned in the item description. Finally, Attunity Connect calls the function to signal the end of the list operation (`GDB_ITEM_LIST_CLOSE_`) and allow the driver to release any resources that it has allocated for the list.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** `GDB_STATUS (*GDB_FNC_GET_ITEM_LIST) (`  
    `GDB_ITEM_DESC **item_desc,`  
    `GDB_LIST_REQUEST request,`  
    `GDB_HANDLE gdb_handle,`  
    `GDB_ITEM_TYPE item_type,`  
    `char *mask);`

Parameter	Usage Description
item_desc	Output A pointer to a GDB_ITEM_DESC structure with the name and description of an item from the data source, such as a table or procedure.
request	Input The type of request needed in the list operation, either GDB_ITEM_LIST_OPEN_, GDB_ITEM_LIST_GET_ or GDB_ITEM_LIST_CLOSE_.
gdb_handle	Input The connection handle to a particular data source, as returned by the driver from the CONNECT function. The driver can assume that Attunity Connect does not have more than one list open for each connection at any given time.
item_type	Input The type of item to be listed: GDB_ITEM_TABLE_ GDB_ITEM_SP GDB_ITEM_VIEW GDB_ITEM_SYNONYM GDB_ITEM_ALIAS GDB_ITEM_LOCAL_TEMPORARY GDB_ITEM_GLOBAL_TEMPORARY GDB_ITEM_SYSTEM_TABLE_  The values can be OR'd together, indicating that you want the returned list for multiple types, such as system tables and views. For details about GDB_ITEM_TYPE, see "type" on page 268. This parameter must be specified for the first call to this function.
mask	Input A mask with wildcards on the required table names. If the driver ignores this mask, Attunity Connect uses the mask to filter out the tables that do not match it. Attunity Connect supports the following wildcards: <b>asterisk (*)</b> – multiple characters. <b>question mark (?)</b> – a single character.

## The DESCRIBE\_TABLE Function

**Assignment** Specify the name of this function as the `io_describe_table` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 259.)

**Description** Attunity Connect calls this function to obtain the full metadata description of a table.

**Status** Required for a non-ADD driver (where the driver supplies its own metadata to Attunity Connect).

Otherwise, the LOAD function should issue the macro GDB\_SET\_ADD and the driver should not implement this function.

**Comments** The driver should not assume that Attunity Connect calls this function before a table is referenced, because the user may decide to cache the data source's metadata in ADD in order to improve performance (see the description of the LOCAL\_COPY utility in the *Attunity Connect Reference*).

Likewise, the driver should not assume that Attunity Connect called the GET\_ITEM\_LIST function before calling DESCRIBE\_TABLE. If Attunity Connect knows the name of a particular table whose metadata it needs, it may simply call the DESCRIBE\_TABLE function for that table rather than first getting a list of all tables belonging to the driver.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDB_FNC_DESCRIBE_TABLE) (
    GDB_HANDLE gdb_handle,
    char *table_name,
    TABLE_DA **table_da);
```

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
table_name	Input	The name of the table for which metadata is requested.
table_da	Output	A TABLE_DA structure allocated by the driver, including all the metadata Attunity Connect needs to know about the table. The driver can free the memory allocated for the TABLE_DA structure as needed, either before getting metadata for another table or at the DISCONNECT function.

## The DESCRIBE\_SP Function

**Assignment** Specify the name of this function as the `io_describe_sp` member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to obtain the full metadata description of a stored procedure.

**Status** Required for a non-ADD driver, that is, the driver supplies its own metadata to Attunity Connect.

Otherwise, the LOAD function should issue the macro GDB\_SET\_ADD and the driver should not implement this function.

**Comments** The driver should not assume that Attunity Connect calls this function before a procedure is referenced, because the user may decide to cache the data source's metadata in an ADD in order to improve performance (see the description of the LOCAL\_COPY utility in the *Attunity Connect Reference*).

Likewise, the driver should not assume that Attunity Connect called the GET\_ITEM\_LIST function before calling DESCRIBE\_SP. If Attunity Connect knows the name of a particular procedure whose metadata it needs, it may simply call the DESCRIBE\_SP function for that procedure rather than first getting a list of all procedures belonging to the driver.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_DESCRIBE\_SP) ( GDB\_HANDLE gdb\_handle, char \*sp\_name, TABLE\_DA \*\*table\_da);

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
sp_name	Input	The name of the stored procedure.
table_da	Output	A TABLE_DA structure allocated by the driver, including all the metadata Attunity Connect needs to know about the procedure. The driver can free the memory allocated for the TABLE_DA structure as needed, either before getting metadata for another procedure or at the DISCONNECT function.

## The GET\_EXTENDED\_METADATA Function

**Assignment** Specify the name of this function as the `io_get_extended_metadata` member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to enable a driver to manipulate or augment metadata based on information from the physical file.

**Status** Optional for both ADD and non-ADD drivers.

**Comments** Some data sources embed metadata in the physical file, typically information related to indexes, cardinality, and record size. Attunity Connect does not store such details in the standard ADD definition of the file. If you use ADD to store metadata for a data driver, this function allows you to merge “extended” metadata from the physical file with ADD metadata.

For a non-ADD driver, the driver is responsible for populating the necessary metadata. This can be done entirely in the DESCRIBE\_TABLE function (see page 40), or the “extended” metadata elements can be retrieved and stored using this function.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDB_FNC_GET_EXTENDED_METADATA) (
    GDB_HANDLE gdb_handle,
    TABLE_DA *table_da,
    long *table_da_changed);
```

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
table_da	Input/O Output	A TABLE_DA structure populated with ADD metadata, which can be changed.
table_da_changed	Output	A flag indicating whether anything has changed in the TABLE_DA structure: <ul style="list-style-type: none"> <li>■ 0 if not changed.</li> <li>■ non-zero if changed.</li> </ul>

## The GET\_FILE\_INFO Function

**Assignment** Specify the name of this function as the `io_get_file_info` member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to enable a driver to return information about a file.

**Status** Optional for both ADD and non-ADD drivers.

**Comments** For a description of possible return values, see "GDB\_STATUS" on page 305. The GDB\_FILE\_INFO structure also includes a status element, containing one of the following values:

```
GDB_FILE_INFO_OK_
GDB_FILE_INFO_UNSUPPORTED_DRIVER_
GDB_FILE_INFO_FILE_NOT_FOUND_
GDB_FILE_INFO_UNKNOWN_
GDB_FILE_INFO_SYSTEM_ERROR_
```

**Prototype** GDB\_STATUS (\*GDB\_FNC\_GET\_FILE\_INFO) (

```
GDB_FILE_INFO gdb_file_info);
```

Parameter	Usage	Description
gdb_file_info	Input/O utput	A GDB_FILE_INFO structure populated with the names of the driver and file for which information is requested.

## Retrieving Data

This section describes the following functions:

- The OPEN Function
- The SET\_BUFFER Function
- The SET\_INDEX Function
- The SET\_FILTER Function
- The SET\_FILTER\_PARAMS Function
- The SET\_PARAMS Function
- The SET\_BM Function
- The CONVERT\_BM\_DATA Function
- The FETCH Function
- The CLOSE Function
- The REWIND Function

### The OPEN Function

**Assignment** Specify the name of this function as the `io_open` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to obtain a new stream on a given table.

**Status** Required.

**Comments** All data retrieval and data manipulation operations require a stream in which to work. The driver must be able to open multiple streams on the same table. The streams should be de-coupled so that an operation on one stream has no effect on the other stream.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_OPEN) (

- GDB\_STREAM \*io\_stream,
- GDB\_HANDLE gdb\_handle,
- TABLE\_DA \*table\_da,
- GDB\_OPEN\_MODE mode);

Parameter	Usage	Description
io_stream	Output	The handle identifying the new stream. Typically the driver returns a pointer to an internal structure holding any stream-specific information that the driver needs.
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
table_da	Input	A TABLE_DA structure allocated by the driver, including all the metadata Attunity Connect needs to know about the table. This parameter makes it easier for the driver to implement the various operations, especially if the driver is an ADD driver, or if LOCAL_COPY was used to cache metadata. The driver can assume that this pointer is valid for the duration of the connection.
mode	Input	The access type to the table, either GDB_OPEN_READ_ONLY_ or GDB_OPEN_READ_WRITE_. Drivers that support transactions may safely assume that in the context of a read-only transaction, mode is always GDB_OPEN_READ_ONLY_.

## The SET\_BUFFER Function

**Assignment** Specify the name of this function as the `io_set_buffer` member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to assign an output buffer (and column list) to a stream. The buffer is the means by which Attunity Connect and the driver communicate, allowing data to be moved as needed to satisfy user requests.

**Status** Required.

**Comments** The buffer assignment is assumed to be in effect until the next SET\_BUFFER call or until the stream is closed.

If the driver does not support column lists, the driver may safely assume that the buffer includes all columns; in this case, Attunity Connect always creates a complete buffer when adding a new record. For details, see "GDB\_M\_SUPPORT\_COLUMN\_LIST\_" on page 295 and "GDB\_M\_INSERT\_ALL\_COLUMNS\_" on page 295.

The possible return values from this API are GDB\_OK\_ and GDB\_NOT\_OK\_. (See "GDB\_STATUS" on page 305.)

**Prototype** GDB\_STATUS (\*GDB\_FNC\_SET\_BUFFER) (  
    GDB\_STREAM io\_stream,  
    GDB\_BUFFER \*io\_buffer);

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
io_buffer	Input	A GDB_BUFFER descriptor for any data retrieval or data manipulation operation that follows. The driver may assume that the buffer pointer remains valid until the next SET_BUFFER call, or until the stream is closed (CLOSE function).

## The SET\_INDEX Function

**Assignment** Specify the name of this function as the `io_set_index` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to set a specific index range for accessing a table in a given stream.

**Status** Required if the driver exposes index metadata for its tables.

**Comments** The Attunity Connect calls SET\_INDEX if the Optimizer chose index access as the optimal strategy. (For more information, see "Query Processor and Optimizer" in the *Attunity Connect Reference*.) All FETCH calls following a SET\_INDEX call should return rows that match the given range.

- ❖ When performing a sequential read of a table, Attunity Connect does not necessarily call SET\_INDEX on the stream; however, if there is an ORDER BY clause that can be satisfied by using an index, Attunity Connect calls SET\_INDEX for optimal performance.

For drivers that support filter expressions on tables, Attunity Connect first calls SET\_FILTER (and SET\_FILTER\_PARAMS, if supported), and then calls SET\_INDEX. This behavior differs if you set the driver

attributes GDB\_M\_FILTER\_WO\_INDEX\_ or  
GDB\_M\_FILTER\_WO\_SEGMENTS\_:

- If the driver supports GDB\_M\_FILTER\_WO\_INDEX\_, Attunity Connect does not call SET\_INDEX since all index values come from the filter tree hierarchy.
- If the driver supports GDB\_M\_FILTER\_WO\_SEGMENTS\_, Attunity Connect calls SET\_INDEX once for each distinct index value. The call to SET\_FILTER allows the driver to process any additional search conditions.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_SET\_INDEX) (

```
GDB_STREAM io_stream,
long index_number,
GDB_INDEX_RELATION index_relation,
GDB_BUFFER *start_buffer,
GDB_BUFFER *end_buffer);
```

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
index_number	Input	The ordinal number into the table_da->index array for this table. Index numbers start at zero.
index_relation	Input	The type of index operation: GDB_EQUAL_ – all rows matching an index value. GDB_GREATER_EQUAL_ – all rows from the index value to the end of file. If the index returns data in descending order, matching records for this type of operation are “less than” or equal to the specified index value. The driver is required to implement both types of index operations.

Parameter	Usage	Description
start_buffer	Input	A buffer descriptor (GDB_BUFFER) with the values of the index segment columns. SET_INDEX may be called with a partial index, but only trailing segments of the index may be omitted. No wildcards are supplied. If the WHERE clause includes an upper limit or Attunity Connect uses the index for ordering only, the start_buffer includes the minimum values for every segment of the index.
end_buffer	Input	A buffer descriptor (GDB_BUFFER) with the index values marking the end of the required range (for example, in a WHERE clause using the BETWEEN keyword to mark the range). If the driver ignores this parameter and returns values greater than the end_buffer values, Attunity Connect recognizes this and conclude the end-of-stream. The driver can use this parameter to implement the range processing more efficiently.

## The SET\_FILTER Function

**Assignment** Specify the name of this function as the `io_set_filter` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 260.)

**Description** Attunity Connect calls this function once before fetching the first row for a given stream. For a driver that supports filter expressions, this routine provides access to the filter expression prior to processing it.

**Status** Required if the driver supports filter expressions (that is, the LOAD function issues the macro `GDB_SET_SUPPORT_FILTERS`).

**Comments** Additional driver attributes (`GDB_M_FILTER_WO_INDEX_`, `GDB_M_FILTER_WO_SEGMENTS_`) affect the filters passed to the driver.

The WHERE clause sets the search conditions for filtering rows in a SELECT, UPDATE, or DELETE statement. Filter expressions employ comparison operators, logical operators, and other common SQL keywords and options. When referring to columns in the table, filter expressions may include non-index columns to identify the subset of desired rows.

If the driver does not support filter expressions or certain search conditions (such as BETWEEN), Attunity Connect performs the necessary filtering after the set has been retrieved by the driver. This

may be less efficient than having the driver return a smaller subset of rows, but it ensures that the user sees only the requested data.

The filter passed down to the driver includes only simple expressions – complex filters using functions are **not** passed to the driver.

The filter is passed to the driver information only. The driver is not committed to implement all or any of the filter nodes. Attunity Connect always filters the rows returned by the driver so that correctness is guaranteed.

Some expressions are simplified when passed to the filter. For example, an expression like "col1 like 'a%" is sent to the driver as "col1>='a' and col2<'ab'".

For a description of possible return values, see "GDB\_STATUS" on page 305.

```
Prototype GDB_STATUS (*GDB_FNC_SET_FILTER) (
    GDB_STREAM io_stream,
    long n_filter_tree,
    long *filter_tree,
    long n_filter_consts,
    GDB_VAL_DESC **filter_consts);
```

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
n_filter_tree	Input	Number of entries in filter tree array.
filter_tree	Input	Filter tree hierarchy. For details, see "GDB_FTREE_NODE" on page 284.
n_filter_consts	Input	Number of entries in filter constants array.
filter_consts	Input	Array of filter constants.

## Parsing a Filter Expression

A valid filter expression can include one or more conditions, with AND or OR between them. The components of a basic condition can be specified with one of the following formats:

- *TableColumn Operator Constant*
- *TableColumn Operator TableColumn*
- *TableColumn Operator Parameter*

A filter expression must be parsed to produce a hierarchical tree representation of these components based on the order of execution

within the expression. The order of execution is determined by the following rules:

- When all operators in an expression are of the same level, the order of execution is left to right.
- When more than one logical operator is used in a statement, AND is evaluated before OR.
- Parentheses can be used to change the order of execution.

The root node of the tree is the operator with the highest precedence. Branching from this node are its related components, which may be either simple operands (column, constant, or parameter) or conditional expressions. The filter tree hierarchy follows the leftmost path of operator subnodes, until it reaches a simple operand. The parser then returns to the previous operator node to process the right-hand portion of that condition, traversing down to a simple operand in the same manner. This left-to-right parsing continues until all components of the filter expression have been reduced to simple operators and operands.

To help you parse a filter expression and build the hierarchical filter tree array in the data driver, Attunity Connect supplies a support function, TRAVERSE\_FILTER. (See page 170.)

## The SET\_FILTER\_PARAMS Function

**Assignment** Specify the name of this function as the `io_set_filter_params` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to set parameter values for a previously set filter expression, before calling SET\_INDEX.

**Status** Required if the driver supports parameters in filter expressions (that is, the LOAD function issues the macro `GDB_SET_SUPPORT_FILTER_PARAMS`).

**Comments** This function is called separately from the SET\_FILTER call that processes the array of constants used in the expression. This allows the driver to efficiently filter data by first using non-variable data (constants) and then using parameters to further reduce the selected subset of rows.

If a driver does not support parameters, Attunity Connect calls SET\_FILTER followed by SET\_INDEX for each variable setting, rather than calling SET\_FILTER\_PARAMS.

For a description of possible return values, see "GDB\_STATUS" on page 305.

---

**Prototype** GDB\_STATUS (\*GDB\_FNC\_SET\_FILTER\_PARAMS) (

```
GDB_STREAM io_stream,
long n_params,
GDB_VAL_DESC **filter_params);
```

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
n_params	Input	Number of filter parameters in array.
filter_params	Output	Array of filter parameters.

## The SET\_PARAMS Function

**Assignment** Specify the name of this function as the **io\_set\_params** member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to set parameters for a stored procedure accessed in a given stream. Attunity Connect also calls this function to set parameters for a command that yields a rowset.

**Status** Required if the driver supports parameters for its stored procedures and/or commands.

**Comments** SET\_PARAMS must be called with values for all parameters referenced by the procedure or command.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_SET\_PARAMS) (

```
GDB_STREAM io_stream,
GDB_BUFFER *params);
```

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
params	Input	A buffer descriptor (GDB_BUFFER) with the values of the parameters for the stored procedure or command.

## The SET\_BM Function

**Assignment** Specify the name of this function as the **io\_set\_bm** member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 260.)

**Description** Attunity Connect calls this function when it needs to re-read a row that was previously read.

**Status** Required whenever bookmarks are needed. That is, in the following cases:

- The driver allows read-write operations.
- The driver supports arrays.

**Comments** Bookmarks uniquely identify rows. The FETCH following a SET\_BM call should return the row pointed to by the given bookmark. The driver can assume that it calls the FETCH function more than once after a SET\_BM call.

The bookmark is not guaranteed to be aligned. This is important for alignment sensitive platforms, such as UNIX platforms, since on these platforms you cannot simply deference a cast bookmark pointer and use its value. You must use memcpy to/from a local variable before setting or using the value of the bookmark.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_SET\_BM) (  
    GDB\_STREAM io\_stream,  
    GDB\_BOOKMARK bm,  
    GDB\_LOCK\_MODE lock\_mode);

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
bm	Input	The bookmark of the requested row. The driver may assume that Attunity Connect obtained this bookmark from the driver in a previous FETCH call. Bookmarks are assumed to be independent of the stream in which they were read.  Attunity Connect pre-allocates this parameter. The bookmark size is either defined for all tables of the driver using the GDB_SET_BOOKMARK_SIZE macro, or on a table by table basis using table_da->bookmark_size. <ul style="list-style-type: none"><li>❖ The driver must check whether bm is not null, since the Attunity Connect query processor does not always require bookmarks.</li></ul>
lock_mode	Input	The desired lock status of the row when fetched, either GDB_NO_LOCK_ or GDB_LOCK_.

## The CONVERT\_BM\_DATA Function

**Assignment** Specify the name of this function as the `io_convert_bm_data` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 261.)

**Description** Attunity Connect calls this function when it needs to convert the bookmark data from or to a string representation.

**Status** Required if the driver supports arrays.

**Comments** Attunity Connect handles arrays by converting the array data into a virtual table (a table that resides in memory). If the driver uses the ADD for metadata, you must specify extended metadata to use virtual tables for the custom data source.

To expose a selected column as an array when the driver supplies its own metadata, populate the "dimension\_1" element of the `COLUMN_DA` structure.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDB_FNC_CONVERT_BM_DATA) (
    GDB_STREAM io_stream,
    GDB_BM_CONVERT_MODE mode,
    GDB_BOOKMARK bm,
    char * str_bm);
```

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
mode	Input	The type of conversion needed, either <code>GDB_BM_FROM_STRING_</code> or <code>GDB_BM_TO_STRING_</code> .
bm	Input/O Output	The bookmark of the requested row.
str_bm	Input/O Output	The string representation of the bookmark, used to connect the parent to its array members. String representation is limited to 64 bytes, null terminated.

## The FETCH Function

**Assignment** Specify the name of this function as the `io_fetch` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to retrieve one row from the stream.

**Status** Required.

**Comments** The driver should return the row according to the buffer descriptor (GDB\_BUFFER) that was set by calling the SET\_BUFFER function.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_FETCH) ( GDB\_STREAM io\_stream, GDB\_BOOKMARK bm);

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
bm	Input/O utput	<p>The bookmark identifying the returned row. The bm parameter is a pointer to a buffer that Attunity Connect allocates according to the bookmark size specified as part of the table's metadata or the driver's attributes. If bookmarks are not used, the bm parameter is null.</p> <ul style="list-style-type: none"><li>❖ Bookmarks are returned by the driver's FETCH function. The "bm" parameter is pre-allocated by Attunity Connect. The driver must check whether bm is not null, since the Query Processor does not always require bookmarks.</li><li>❖ The bookmark size is either defined for all tables of the driver that uses the GDB_SET_BOOKMARK_SIZE macro, or on a table by table basis using table_da-&gt;bookmark_size.</li><li>❖ The bookmark is not guaranteed to be aligned. This is important for alignment sensitive platforms, such as UNIX platforms, since on these platforms you cannot simply deference a cast bookmark pointer and use its value. You must use memcpy to/from a local variable before setting or using the value of the bookmark.</li></ul>

## The CLOSE Function

**Assignment** Specify the name of this function as the **io\_close** member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to terminate an existing stream.

**Status** Required.

**Comments** The driver should free any resources associated with this stream.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_CLOSE) (  
    GDB\_STREAM io\_stream);

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.

## The REWIND Function

**Assignment** Specify the name of this function as the `io_rewind` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 260.)

**Description** Attunity Connect calls this function to rewind a stream back to the first record in the stream.

**Status** Required.

**Comments** A call to FETCH following a call to REWIND should fetch the same record as a call to FETCH following a call to OPEN. This allows Attunity Connect to reuse the same stream rather than opening additional streams.

The REWIND function operates on an active stream and retains the given index and buffers. A subsequent call to FETCH uses the given index relation and returns the first row in the given index order.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_REWIND) (  
    GDB\_STREAM io\_stream);

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.

## Updating Data

This section describes the following functions:

- The ADD\_ROW Function
- The UPDATE\_ROW Function
- The DELETE\_ROW Function
- The LOCK\_ROW Function
- The UNLOCK\_ROW Function

### The ADD\_ROW Function

**Assignment** Specify the name of this function as the `io_add_row` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 261.)

**Description** Attunity Connect calls this function to insert a new row in a table.

The values for the table columns were set to the current stream buffer (using `SET_BUFFER`) before calling this function. The driver may assume that columns declared as not nullable are part of the given buffer. Drivers that do not support column lists always receive a full buffer.

**Status** Required if the driver supports read-write I/O processing (that is, the `LOAD` function issues the macro `GDB_SET_SUPPORT_UPDATE`).

**Comments** For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** `GDB_STATUS (*GDB_FNC_ADD_ROW) (`  
    `GDB_STREAM io_stream,`  
    `GDB_BOOKMARK bm);`

Parameter	Usage	Description
<code>io_stream</code>	Input	The handle identifying the stream, as returned by the driver from the <code>OPEN</code> function.
<code>bm</code>	Output	The bookmark identifying the newly added row. The <code>bm</code> parameter is a pointer to a buffer that Attunity Connect allocates according to the bookmark size specified as part of the table's metadata or the driver's attributes.

### The UPDATE\_ROW Function

**Assignment** Specify the name of this function as the `io_update_row` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 261.)

**Description** Attunity Connect calls this function to update one row identified by the given bookmark.

**Status** Required if the driver supports read-write I/O processing (that is, the LOAD function issues the macro GDB\_SET\_SUPPORT\_UPDATE).

**Comments** The new values reside in the current stream buffer (GDB\_BUFFER). Drivers that do not support column lists always receive a full buffer.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDB_FNC_UPDATE_ROW) (
    GDB_STREAM io_stream,
    GDB_BOOKMARK bm);
```

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
bm	Input/O utput	A pointer to a bookmark buffer holding the bookmark of the row to be updated. In some cases, such as changing a unique index value, updating the row affects the bookmark, so the driver may return a different bookmark in the given bookmark buffer.

## The DELETE\_ROW Function

**Assignment** Specify the name of this function as the `io_delete_row` member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 261.)

**Description** Attunity Connect calls this function to delete one row identified by its bookmark.

**Status** Required if the driver supports read-write I/O processing (that is, the LOAD function issues the macro GDB\_SET\_SUPPORT\_UPDATE).

**Comments** For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDB_FNC_DELETE_ROW) (
    GDB_STREAM io_stream,
    GDB_BOOKMARK bm);
```

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.

Parameter	Usage	Description
bm	Input	A pointer to a bookmark buffer holding the bookmark of the row to be deleted.

## The LOCK\_ROW Function

<b>Assignment</b>	Specify the name of this function as the <code>io_lock_row</code> member of the <code>GDB_DB_FUNCTIONS</code> structure. (For details, see page 261.)	
<b>Description</b>	Attunity Connect calls this function to lock one row identified by its bookmark.	
<b>Status</b>	Required if the driver supports read-write I/O processing (that is, the LOAD function issues the macro <code>GDB_SET_SUPPORT_UPDATE</code> ).	
<b>Comments</b>	This call may occur as a result of a user (client) action, particularly when the driver supports delayed locking in transaction processing.	
	For a description of possible return values, see "GDB_STATUS" on page 305.	
<b>Prototype</b>	<pre>GDB_STATUS (*GDB_FNC_LOCK_ROW) (     GDB_STREAM io_stream,     GDB_BOOKMARK bm);</pre>	
Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
bm	Input	A pointer to a bookmark buffer holding the bookmark of the row to be locked.

## The UNLOCK\_ROW Function

<b>Assignment</b>	Specify the name of this function as the <code>io_unlock_row</code> member of the <code>GDB_DB_FUNCTIONS</code> structure. (For details, see page 261.)
<b>Description</b>	Attunity Connect calls this function to unlock one row identified by its bookmark.
<b>Status</b>	Required if the driver supports read-write I/O processing (that is, the LOAD function issues the macro <code>GDB_SET_SUPPORT_UPDATE</code> ).
<b>Comments</b>	This call may occur as a result of a user (client) action, particularly when the driver supports delayed locking in transaction processing.
	For a description of possible return values, see "GDB_STATUS" on page 305.

---

```
Prototype GDB_STATUS (*GDB_FNC_UNLOCK_ROW) (
    GDB_STREAM io_stream,
    GDB_BOOKMARK bm);
```

Parameter	Usage	Description
io_stream	Input	The handle identifying the stream, as returned by the driver from the OPEN function.
bm	Input	A pointer to a bookmark buffer holding the bookmark of the row to be unlocked.

## Managing Transactions

This section describes the following functions:

- The START\_TRANSACTION Function
- The COMMIT\_TRANSACTION Function
- The ROLLBACK\_TRANSACTION Function
- The START\_DISTRIBUTED\_TRANSACTION Function (*for future use*)
- The PREPARE\_COMMIT\_TRANSACTION Function (*for future use*)
- The RECOVER\_TRANSACTION Function (*for future use*)

### The START\_TRANSACTION Function

**Assignment** Specify the name of this function as the `io_start_transaction` member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 261.)

**Description** Attunity Connect calls this function to start a transaction for the driver (for example, when a user executes an SQL BEGIN statement).

**Status** Required if the driver supports transaction processing (that is, the LOAD function issues the macro GDB\_SET\_SUPPORT\_TRANS).

**Comments** Attunity Connect assumes the driver implements a read-committed isolation level.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_START\_TRANSACTION) (  
    GDB\_HANDLE gdb\_handle, GDB\_TRANS\_MODE tran\_mode);

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from CONNECT.
tran_mode	Input	The transaction mode, specified as either GDB_TRANS_READ_ONLY or GDB_TRANS_READ_WRITE depending on the type of operations to be performed within the transaction.

## The COMMIT\_TRANSACTION Function

**Assignment** Specify the name of this function as the `io_commit_transaction` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 261.)

**Description** Attunity Connect calls this function to commit an active transaction for the driver (for example, when a user executes an SQL COMMIT statement).

**Status** Required if the driver supports transaction processing (that is, the LOAD function issues the macro `GDB_SET_SUPPORT_TRANS`).

**Comments** For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_COMMIT\_TRANSACTION) (  
    GDB\_HANDLE gdb\_handle);

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from CONNECT.

## The ROLLBACK\_TRANSACTION Function

**Assignment** Specify the name of this function as the `io_rollback_transaction` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 261.)

**Description** Attunity Connect calls this function to rollback an active transaction for the driver (for example, when a user executes an SQL ROLLBACK statement).

**Status** Required if the driver supports transaction processing (that is, the LOAD function issues the macro `GDB_SET_SUPPORT_TRANS`).

**Comments** For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_ROLLBACK\_TRANSACTION) ( GDB\_HANDLE gdb\_handle);

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from CONNECT.

## The START\_DISTRIBUTED\_TRANSACTION Function

❖ *For future use:*

This function will be implemented as part of distributed, two-phase commit transaction support, in an upcoming version of the Developer SDK.

**Assignment** Specify the name of this function as the `io_start_distributed_transaction` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 261.)

## The PREPARE\_COMMIT\_TRANSACTION Function

❖ *For future use:*

This function will be implemented as part of distributed, two-phase commit transaction support, in an upcoming version of the Developer SDK.

**Assignment** Specify the name of this function as the `io_prepare_commit_transaction` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 261.)

## The RECOVER\_TRANSACTION Function

❖ *For future use:*

This function will be implemented as part of distributed, two-phase commit transaction support, in an upcoming version of the Developer SDK.

**Assignment** Specify the name of this function as the `io_recover_transaction` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

## Data Definition

This section describes the following functions:

- The CREATE\_TABLE Function
- The CREATE\_INDEX Function
- The DROP\_TABLE Function
- The DROP\_INDEX Function (*for future use*)

## The CREATE\_TABLE Function

**Assignment** Specify the name of this function as the `io_create_table` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

**Description** Attunity Connect calls this function to create a new table (for example, when the user executes an SQL CREATE TABLE statement).

**Status** Optional.

**Comments** See below for a list of data types that you can include in a table defined with the CREATE\_TABLE function.

If you are providing metadata as part of the driver, you must also add whatever is needed in the backend database when the driver's CREATE\_TABLE function is called. This ensures that the next time Attunity Connect asks the driver for metadata, the results include the added table. Attunity Connect adds the table to the cached metadata within the current session.

For a data driver that uses ADD, Attunity Connect physically creates the ADD metadata after calling the driver's CREATE\_TABLE function.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** `GDB_STATUS (*GDB_FNC_CREATE_TABLE) (`  
    `GDB_HANDLE gdb_handle,`  
    `TABLE_DA *table_da);`

Parameter	Usage	Description
<code>gdb_handle</code>	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
<code>table_da</code>	Input	A TABLE_DA structure allocated by the driver, containing a description of the table's attributes and fields that the driver should create. <ul style="list-style-type: none"><li>❖ The TABLE_DA structure can include index definitions, but this function ignores these specifications. All indexes are created using the CREATE_INDEX function.</li></ul>

**CREATE\_TABLE Data Types** The GDB API supports the standard SQL data types in the CREATE\_TABLE function.

SQL Data Type	Attunity Connect Data Type Code	ADD Type
Char[(m)]	DT_TYPE_T_	STRING

Varchar(m)	DT_TYPE_CSTRING_	CSTRING
Float	DT_TYPE_F_	FFLOAT
Double	DT_TYPE_G_	GFLOAT
Tinyint	DT_TYPE_B_	BYTE
Smallint	DT_TYPE_W_	WORD
Integer	DT_TYPE_L_	LONGWORD
Numeric	DT_TYPE_NRO_	NUMSTR_S
Numeric(p[,s])	DT_TYPE_NRO_	NUMSTR_NR
Date	DT_TYPE_ODBC_DATE_	APT_DATE
Time	DT_TYPE_ODBC_TIME_	APT_TIME
Datetime	DT_TYPE_ODBC_TIMESTAMP_	VMS_DATE
Image	<i>Not supported</i>	
Image(m)	<i>Not supported</i>	
Text	<i>Not supported</i>	

## The CREATE\_INDEX Function

**Assignment** Specify the name of this function as the `io_create_index` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

**Description** Attunity Connect calls this function to add a new index to a previously created table (for example, when the user executes an SQL CREATE INDEX statement).

**Status** Optional.

**Comments** If you are providing metadata as part of the driver, you must also add whatever is needed in the backend database when the driver's `CREATE_INDEX` function is called. This ensures that the next time Attunity Connect asks the driver for metadata, the results include the added index. Attunity Connect adds the index to the cached metadata within the current session.

For data drivers that use ADD, Attunity Connect physically creates the ADD metadata after calling the driver's `CREATE_INDEX` function.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** `GDB_STATUS (*GDB_FNC_CREATE_INDEX) (`  
`GDB_HANDLE gdb_handle,`

```
TABLE_DA *table_da,  
long index_number);
```

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
table_da	Input	A TABLE_DA structure allocated by the driver, containing a full description of the table, including the new index to be created.
index_number	Input	The ordinal number into the table_da->index array that describes the index to be created. Index numbers start at zero.

## The DROP\_TABLE Function

**Assignment** Specify the name of this function as the `io_drop_table` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

**Description** Attunity Connect calls this function to delete an existing table (for example, when a user executes an SQL `DROP TABLE` statement).

**Status** Optional.

**Comments** If you are providing metadata as part of the driver, you must also delete whatever exists in the backend database when the driver's `DROP_TABLE` function is called. This ensures that the next time Attunity Connect asks the driver for metadata, the results do not include the dropped table. Attunity Connect removes the table from the cached metadata within the current session.

For data drivers that use ADD, Attunity Connect physically deletes the ADD metadata after calling the driver's `DROP_TABLE` function.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** `GDB_STATUS (*GDB_FNC_DROP_TABLE) (`  
`GDB_HANDLE gdb_handle,`  
`char *table_da);`

---

### Parameter Usage Description

gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
------------	-------	---

**Parameter Usage Description**

table_da	Input	A "TABLE_DA" structure allocated by the driver, containing a full description of the table to be deleted.
----------	-------	---

## The DROP\_INDEX Function

❖ *For future use:*

Attunity Connect calls this function to drop an existing index (for example, when the user executes an SQL DROP INDEX command). This function will be implemented in an upcoming version of the Developer SDK.

**Assignment** Specify the name of this function as the `io_drop_index` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

## BLOB Support

This section describes the following functions:

- The BLOB\_OPEN Function
- The BLOB\_READ Function
- The BLOB\_WRITE Function
- The BLOB\_SEEK Function (*for future use*)
- The BLOB\_CLOSE Function

## The BLOB\_OPEN Function

**Assignment** Specify the name of this function as the `io_blob_open` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

**Description** Attunity Connect calls this function to establish a BLOB stream for both reading and writing BLOBS.

**Status** Required if the driver supports BLOBS (see "GDB\_COL\_M\_BLOB\_" on page 301).

**Comments** The BLOB\_OPEN function should return a `blob_handle` which is used by all the other BLOB functions to identify the BLOB stream.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDB_FNC_BLOB_OPEN) (
    GDB_HANDLE gdb_handle,
    GDB_BLOB *blob_handle,
    TABLE_DA *table_da,
```

```
long column_number,  
GDB_BOOKMARK bm);
```

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
blob_handle	Output	A handle that identifies this BLOB stream to the driver. Typically the driver returns in this parameter the address of an internal structure that includes all the information the driver requires about the BLOB stream.
table_da	Input	A "TABLE_DA" structure allocated by the driver, containing the metadata of the table associated with the BLOB, for the driver's reference.
column_number	Input	The ordinal number of the BLOB column in the table_da->columns array (see page 273). Column numbers start at zero.
bm	Input	The bookmark of the row from which the BLOB stream needs to be established.

## The BLOB\_READ Function

**Assignment** Specify the name of this function as the **io\_blob\_read** member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 262.)

**Description** Attunity Connect calls this function to read a given number of bytes from the BLOB stream to a pre-allocated buffer.

**Status** Required if the driver supports BLOBS (see "GDB\_COL\_M\_BLOB\_" on page 301).

**Comments** Processing of a large BLOB may require several calls to BLOB\_READ in order to fill the requested size. When the BLOB current position is at the end of the BLOB stream, this function should return a GDB\_END\_OF\_STREAM\_ return code and the actual\_size\_read should be set to 0.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_BLOB\_READ) (  
    GDB\_BLOB blob\_handle,  
    long requested\_size,

```
long *actual_size_read,
void *blob_buffer);
```

Parameter	Usage	Description
blob_handle	Input	The handle of the BLOB stream as returned by BLOB_OPEN.
requested_size	Input	The number of bytes to be read from the BLOB stream to the given buffer.
actual_size_read	Output	The actual number of bytes that the driver put in the supplied buffer.
blob_buffer	Output	The address of a buffer that is allocated by Attunity Connect. The driver must populate this buffer from the BLOB stream.

## The BLOB\_WRITE Function

<b>Assignment</b>	Specify the name of this function as the <b>io_blob_write</b> member of the GDB_DB_FUNCTIONS structure. (For details, see page 262.)	
<b>Description</b>	Attunity Connect calls this function to write a buffer of a given size to the BLOB stream.	
<b>Status</b>	Required if the driver supports blobs and update operations (see "GDB_COL_M_BLOB_" on page 301 and "GDB_M_SUPPORT_UPDATE_" on page 296).	
<b>Comments</b>	The data should be written from the current position in the stream, overwriting existing data.	
	For a description of possible return values, see "GDB_STATUS" on page 305.	
<b>Prototype</b>	GDB_STATUS (*GDB_FNC_BLOB_WRITE) (   GDB_BLOB blob_handle,   long buffer_size,   void *blob_buffer);	
<b>Parameter</b>	<b>Usage</b>	<b>Description</b>
blob_handle	Input	The handle of the BLOB stream as returned by BLOB_OPEN.
buffer_size	Input	The number of bytes in the given buffer that are to be written to the BLOB stream.
blob_buffer	Input	The address of the buffer where the data is written.

## The BLOB\_SEEK Function

- ❖ *For future use:*

Attunity Connect calls this function to set the current position in the BLOB stream at a specific offset from the beginning of the stream. This function will be implemented in an upcoming version of the Developer SDK.

**Assignment** Specify the name of this function as the `io_blob_seek` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

## The BLOB\_CLOSE Function

**Assignment** Specify the name of this function as the `io_blob_close` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

**Description** Attunity Connect calls this function to terminate a given BLOB stream.

**Status** Required if the driver supports BLOBS (see "GDB\_COL\_M\_BLOB\_" on page 301).

**Comments** In this function, the driver should free any resources associated with the BLOB stream.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** `GDB_STATUS (*GDB_FNC_BLOB_CLOSE) (`  
                  `GDB_BLOB blob_handle);`

Parameter	Usage	Description
<code>blob_handl</code>	Input	The handle of the BLOB stream as returned by <code>BLOB_OPEN</code> .

## SQL and non-SQL Command Support

This section describes the following functions:

- The PREPARE\_COMMAND Function
- The DESCRIBE\_COMMAND Function
- The EXECUTE\_IMMEDIATE Function
- The EXECUTE\_WITH\_PARAMS Function
- The FREE\_COMMAND Function
- The GET\_DATE\_LITERAL Function

## The PREPARE\_COMMAND Function

**Assignment** Specify the name of this function as the `io_prepare_command` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

**Description** Attunity Connect calls this function to obtain a cursor handle associated with the command text.

**Status** Required if the driver supports commands (see "GDB\_M\_SUPPORT\_PASSTHRU\_COMMAND\_" on page 297 and "GDB\_M\_SUPPORT\_SQL\_" on page 300).

**Comments** If the command includes one or more parameters and yields a rowset, Attunity Connect calls the `SET_PARAMS` function (page 51) along with `PREPARE_COMMAND`. If the command includes parameters but does not result in a rowset, Attunity Connect simply calls `EXECUTE_WITH_PARAMS` (page 71).

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDB_FNC_PREPARE_COMMAND) (
    GDB_HANDLE gdb_handle,
    GDB_COMMAND * gdb_command,
    char *command_text);
```

Parameter	Usage	Description
<code>gdb_handle</code>	Input	The connection handle to a particular data source, as returned by the driver from the <code>CONNECT</code> function.
<code>gdb_command</code>	Output	A cursor to the command statement.
<code>command_text</code>	Input	Text to be processed by driver.

## The DESCRIBE\_COMMAND Function

**Assignment** Specify the name of this function as the `io_describe_command` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 262.)

**Description** Attunity Connect calls this function to obtain metadata for the rowset (table\_da) resulting from the command.

**Status** Required if the driver supports commands (see "GDB\_M\_SUPPORT\_PASSTHRU\_COMMAND\_" on page 297 and "GDB\_M\_SUPPORT\_SQL\_" on page 300).

**Comments** The `DESCRIBE_COMMAND` function should set the `gdb_command` value in the `TABLE_DA` structure, based on the cursor input value as returned by `PREPARE_COMMAND`. If the `gdb_command` value is not populated

when the OPEN function is called for the resultset of a command, Attunity Connect updates the cursor in the structure and also set the table attribute GDB\_M\_TABLE\_COMMAND\_.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_DESCRIBE\_COMMAND) (

```
    GDB_COMMAND gdb_command,
    TABLE_DA ** table_da);
```

Parameter	Usage	Description
gdb_command	Input	A cursor to the command statement.
table_da	Output	Pointer to a TABLE_DA structure populated with a description of the rowset resulting from the command text.

## The EXECUTE\_IMMEDIATE Function

**Assignment** Specify the name of this function as the `io_execute_immediate` member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 262.)

**Description** Attunity Connect executes a driver command which does not result in a rowset.

**Status** Required if the driver supports commands (see "GDB\_M\_SUPPORT\_PASSTHRU\_COMMAND\_" on page 297 and "GDB\_M\_SUPPORT\_SQL\_" on page 300).

**Comments** A command without a resulting rowset, such as a delete operation, does not need metadata, so Attunity Connect omits calls to PREPARE\_COMMAND and DESCRIBE\_COMMAND when calling EXECUTE\_IMMEDIATE.

The function can return information about the number of rows affected by the command, but cannot return the data from those rows as a rowset.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_EXECUTE\_IMMEDIATE) (

```
    GDB_HANDLE gdb_handle,
```

```
char *command_text,
long *n_affected_rows);
```

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
command_text	Input	Text to be processed by driver.
n_affected_rows	Output	Number of rows affected by command.

## The EXECUTE\_WITH\_PARAMS Function

**Assignment** Specify the name of this function as the `io_execute_with_params` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 263.)

**Description** Attunity Connect executes a driver command that does not result in a rowset but does accept input parameters.

**Status** Optional.

**Comments** This function allows you to reuse a command by supplying parameters for values instead of issuing separate commands with specific literal values. For example, an insert operation can support different entries for the data elements but the underlying command is the same. Attunity Connect calls `PREPARE_COMMAND` once, followed by one or more calls to `EXECUTE_WITH_PARAMS`, as needed.

The parameter values are passed as value descriptors, rather than buffer columns, because there is no way to guarantee the set of data types that will be returned.

The function can return information about the number of rows affected by the command, but cannot return the data from those rows as a rowset.

If this function is not supplied, Attunity Connect converts the parameter values to literals and call `EXECUTE_IMMEDIATE`. (For details, see page 70.)

If the command includes input parameters and yields a rowset, Attunity Connect calls the `SET_PARAMS` function (see page 51) along with `PREPARE_COMMAND` (see page 69).

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_EXECUTE\_WITH\_PARAMS) (

GDB\_COMMAND gdb\_command,  
long n\_params,  
GDB\_VAL\_DESC \* params,  
long \*n\_affected\_rows);

Parameter	Usage	Description
gdb_command	Input	A cursor to the command statement.
n_params	Input	Number of parameters
params	Input	A list of value descriptors for the parameters
n_affected_rows	Output	Number of rows affected by command.

## The FREE\_COMMAND Function

**Assignment** Specify the name of this function as the `io_free_command` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 263.)

**Description** Attunity Connect calls this function to free resources associated with the command.

**Status** Optional but highly recommended, to avoid errors due to conflicts with locked or held resources.

**Comments** For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_FREE\_COMMAND) (

GDB\_COMMAND gdb\_command);

Parameter	Usage	Description
gdb_command	Input	A cursor to the command statement.

## The GET\_DATE\_LITERAL Function

**Assignment** Specify the name of this function as the `io_get_date_literal` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 263.)

**Description** For an SQL driver, Attunity Connect calls this function whenever it has a date value from the data source that needs to be converted to a literal.

**Status** Required if the driver supports SQL commands (see "GDB\_M\_SUPPORT\_SQL\_" on page 300).

**Comments** Since date literal notations vary between databases, the driver must supply a function that produces a literal representation of a given date.

Attunity Connect uses this function for any value whose underlying data type is a date, including user-defined “date” data types (see “[UDT\\_M\\_DATE\\_](#)” on page 303).

For a description of possible return values, see “[GDB\\_STATUS](#)” on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_GET\_DATE\_LITERAL) ( GDB\_HANDLE gdb\_handle, char \*date\_literal, GDB\_VAL\_DESC \* date\_value);

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
date_literal	Input/O	A buffer allocated by Navigator for the output utput
date_value	Input	A value descriptor (GDB_VAL_DESC) for the date

## Chapter Support

This section describes the following functions:

- The DESCRIBE CHAPTER Function
- The OPEN CHAPTER Function
- The SET\_ACTIVE CHAPTER Function

### The DESCRIBE CHAPTER Function

**Assignment** Specify the name of this function as the `io_describe_chapter` member of the GDB\_DB\_FUNCTIONS structure. (For details, see page 263.)

**Description** Attunity Connect calls this function to obtain metadata about a specific chapter in a parent table.

**Status** Required if the driver supports chapters.

**Comments** The metadata for the parent table includes a reference to the chapter at the column level (see “`chapter_table_da`” on page 280). This allows the DESCRIBE CHAPTER function to identify the child chapter and access its metadata as a complete table.

For a description of possible return values, see “[GDB\\_STATUS](#)” on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_DESCRIBE\_CHAPTER) (

```
    GDB_HANDLE gdb_handle,
    TABLE_DA * parent_table_da,
    long column_number,
    TABLE_DA ** chapter_table_da);
```

Parameter	Usage	Description
gdb_handle	Input	The connection handle to a particular data source, as returned by the driver from the CONNECT function.
parent_table_da	Input	A TABLE_DA structure populated with metadata for the parent table.
column_number	Input	Column number of embedded child chapter
table_da	Output	Pointer to a TABLE_DA structure containing the chapter metadata.

## The OPEN CHAPTER Function

**Assignment** Specify the name of this function as the `io_open_chapter` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 263.)

**Description** Attunity Connect calls this function to open a chapter stream.

**Status** Required if the driver supports chapters.

**Comments** The OPEN CHAPTER function is not associated with a specific parent table, but provides a framework for accessing a chapter stream.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** GDB\_STATUS (\*GDB\_FNC\_OPEN CHAPTER) (

```
    GDB_STREAM * chapter_io_stream,
    GDB_STREAM parent_io_stream,
    long parent_chapter_column,
    TABLE_DA * chapter_table_da);
```

Parameter	Usage	Description
chapter_io_stream	Output	The handle identifying the new stream for the chapter (child) table. Typically the driver returns a pointer to an internal structure holding any stream-specific information that the driver needs.
parent_io_stream	Input	The handle identifying the parent stream.

Parameter	Usage	Description
parent_chapter_column	Input	Column number of chapter, embedded in parent table
chapter_table_da	Input	A TABLE_DA structure allocated by the driver, including all the metadata Attunity Connect needs to know about the chapter table. This parameter makes it easier for the driver to implement the various operations.

## The SET\_ACTIVE CHAPTER Function

**Assignment** Specify the name of this function as the `io_set_active_chapter` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 263.)

**Description** Attunity Connect calls this function to set the parent row (the “active” chapter) for a chapter stream.

**Status** Required if the driver supports chapters.

**Comments** The driver can return anything in the `chapter_value`, but after calling this function Attunity Connect must be able to identify the chapter and use it for a `FETCH` of the parent data.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** `GDB_STATUS (*GDB_FNC_SET_ACTIVE_CHAPTER) (`  
`GDB_STREAM chapter_io_stream,`  
`GDB_VAL_DESC * chapter_value);`

Parameter	Usage	Description
chapter_io_stream	Input	The handle identifying the stream for the chapter (child) table, as returned by the <code>OPEN_CHAPTER</code> function.
chapter_value	Input/O utput	Value descriptor used to identify active chapter (parent row) in preparation for <code>FETCH</code> .

## Error Functions

- ❖ All exceptions arising from user-written functions generate an appropriate error message in the log file.

## The GET\_LAST\_ERROR Function

**Assignment** Specify the name of this function as the `io_get_last_error` member of the `GDB_DB_FUNCTIONS` structure. (For details, see page 259.)

**Description** Attunity Connect calls this function when any of the driver functions returns an error code such as `GDB_NOT_OK_`. (See page 305.)

**Status** Optional.

**Comments** Attunity Connect writes the value of the string returned by this function to the log file. The string is also available to the client, so the driver can give the user textual information about the error. Write the driver so that each data source connection can be uniquely identified in the event of an error.

- ❖ The default log file (NAV.LOG, or NAVLOG on Tandem platforms) is located in the TMP directory under the directory where Attunity Connect is installed (on Tandem platforms, in the subvolume where Attunity Connect is installed). Use the `logFile` attribute in the Attunity Connect environment settings to specify a different file or location, as follows: `<debug logFile="pathname" >`.

Under OS/390, the string values are written to the job specified in the `StartupScript` parameter of the `Workspace` section of the daemon configuration.

**Prototype** `char * (*GDB_FNC_GET_LAST_ERROR) (`  
`GDB_HANDLE gdb_handle);`

### Parameter Usage Description

---

`gdb_handle` Input The connection handle to a particular data source, as returned by the driver from `CONNECT`.

---

### Return Value      Usage      Description

Return Value	Usage	Description
<code>&lt;last error text&gt;</code>	Output to log file	A pointer to a string with a description of the last error to occur for this data source.

# Chapter 4    Security Hooks

Access to data and machines via Attunity Connect is controlled by the information in the user profile in the Attunity Connect repository and daemon settings (such as users allowed to access the daemon workspace). Being able to overlay this level of security with user specific requirements is available in the form of security hooks. These security hooks enable users to deploy their own security policies at a number of levels (from simple checking during connection/disconnection to checking SQL operations and tables accessed).

The following security hooks are provided as part of the SDK:

**SESSION-INITIALIZATION** – The check is made when the Attunity Connect session is initialized.

**PASSWORD-PROMPT** – A prompt is always displayed requesting a username and password.

**CONNECTION** – The check is made when a client attempts to connect to a daemon. The check is made against the username supplied by the client.

**QUERY** – The check is made when the SQL is issued as to whether the SQL is allowed or not. For example, SELECT statements might be allowed, but INSERT statements denied.

**TABLE (OBJECT)** – The object in the SQL is checked. For example, a table might be allowed to be accessed but not a stored procedure.

**DISCONNECT** – The check is made when a client attempts to disconnect from a server.

**SESSION-TERMINATION** – The check is made when an attempt is made to shut down the daemon.

## How to Use Security Hooks

Attunity Connect includes a shell security hook program (available on request), which you can use as the basis for a start-up function within the SDK. You can incorporate code, or call other code, to check whether to allow the action or not. Thus, existing security can be incorporated into Attunity Connect. Alternatively, you can write code that is tailored to specific needs that uses the security hooks and incorporate it into Attunity Connect as a start-up function.

For details about start-up functions for Attunity Connect, see "Registering a Startup Function" on page 151.

**Setting Up Security Hooks**

The security hook software must be defined to the ADDON.DEF file used to control SDK functionality. The following example uses the Attunity Connect sechook program (available on request), as follows:

```
[SECHOOK]
TYPE=STARTUP
SHAREABLE-NAME=sechook_register
INIT-FUNCTION=sechook
```

**Monitoring Security Hooks**

Setting gdbTrace to "true" enables tracing the SDK functionality. You can use this setting to test that the security hooks are being called correctly.

**Example**

The following sample program calls the security hooks included in the Attunity Connect SDK. You can use this program as the basis for the security:

```
/* Sample Attunity Connect Security Hook
*/
#include <stdio.h>
#include <string.h>

#include "dbgdb.h"

#ifdef WIN32
#   include <windows.h>
#   define EXPORT_SYMBOL __declspec(dllexport)
#   define ATOMIC_INC(X) InterlockedIncrement(&X)
                  /* ^-- may not work on Win 95 */
#else
#   define EXPORT_SYMBOL
#   define ATOMIC_INC(X) X++
#endif

GDB_HELP_DEFINE;

static SECHOOK_ACTION SecHook(SECHOOK_CTX *sechook_ctx)
{
    char *text;
    static long instance = 0;
    char command[64];

    switch (sechook_ctx->event) {
```

```
case SEHOOK_EVENT_ON_INIT_:
    sehook_ctx->hook_ctx = (void *)ATOMIC_INC(instance);
    GDB_DEBUG_PRINT(GDB_DEBUG_MODE, "SEHOOK: Init(%d)",
                    (long) sehook_ctx->hook_ctx);
    break;

case SEHOOK_EVENT_ON_CONNECT_:
    GDB_DEBUG_PRINT(GDB_DEBUG_MODE,
                    "SEHOOK: Connect(%d), username=%s",
                    (long) sehook_ctx->hook_ctx,
                    sehook_ctx->username);
    break;

case SEHOOK_EVENT_ON_COMMAND_:
    *command = '\0';
    if (sehook_ctx->command & SEHOOK_CMD_SELECT_)
        strcat(command, "SELECT ");
    if (sehook_ctx->command & SEHOOK_CMD_INSERT_)
        strcat(command, "INSERT ");
    if (sehook_ctx->command & SEHOOK_CMD_DELETE_)
        strcat(command, "DELETE ");
    if (sehook_ctx->command & SEHOOK_CMD_UPDATE_)
        strcat(command, "UPDATE ");
    if (sehook_ctx->command & SEHOOK_CMD_CALL_)
        strcat(command, "CALL ");
    if (sehook_ctx->command & SEHOOK_CMD_DDL_)
        strcat(command, "DDL ");
    if (sehook_ctx->command & SEHOOK_CMD_PASSTHRU_)
        strcat(command, "PASSTHRU ");
    if (sehook_ctx->command & SEHOOK_CMD_UNKNOWN_)
        strcat(command, "UNKNOWN ");

    GDB_DEBUG_PRINT(GDB_DEBUG_MODE,
                    "SEHOOK: Command(%d), command=%s",
                    (long) sehook_ctx->hook_ctx,
                    command);
    break;

case SEHOOK_EVENT_ON_OBJECT_:
    switch (sehook_ctx->obj_type) {
        case SEHOOK_OBJTYPE_SERVER_: text = "SERVER"; break;
        case SEHOOK_OBJTYPE_DATABASE_: text = "DATABASE"; break;

        case SEHOOK_OBJTYPE_TABLE_: text = "TABLE"; break;
        case SEHOOK_OBJTYPE_PROCEDURE_: text = "PROCEDURE"; break;
        default: text = "<unknown>"; break;
    }
```

```
        GDB_DEBUG_PRINT(GDB_DEBUG_MODE,
    "SECHOOK: Object(%d), obj_type=%s, datasource=%s, owner=%s,
object=%s",
                (long) sechook_ctx->hook_ctx,
                text, sechook_ctx->datasource, sechook_ctx->owner,
sechook_ctx->object);

        break;

case SECHOOK_EVENT_ON_DISCONNECT_:
    GDB_DEBUG_PRINT(GDB_DEBUG_MODE, "SECHOOK: Disconnect(%d)",
                (long) sechook_ctx->hook_ctx);
        break;

case SECHOOK_EVENT_ON_TERM_:
    GDB_DEBUG_PRINT(GDB_DEBUG_MODE, "SECHOOK: Term(%d)",
                (long) sechook_ctx->hook_ctx);
        break;

case SECHOOK_EVENT_ON_PWD_PROMPT_:
    GDB_DEBUG_PRINT(GDB_DEBUG_MODE, "SECHOOK: PwdPrompt(%d)",
                (long) sechook_ctx->hook_ctx);
        break;
}

return (SECHOOK_ACTION_ALLOW_);
}

/* Registration function */
EXPORT_SYMBOL void sechook_register(GDB_HELP_FUNCTIONS *help)
{
    gdb_help = help;

    GDB_SET_SECHOOK(SecHook);
}
```

Each case "SECHOOK\_EVENT\_ON..." is trapped by Attunity Connect when the event is encountered and a message is entered to the log (if the environment parameter `gdbTrace` is set to "true". For example, the following piece of code writes a message to the log when a connection is made to a database:

```
case SECHOOK_EVENT_ON_CONNECT_:
    GDB_DEBUG_PRINT(GDB_DEBUG_MODE,
    "SECHOOK: Connect(%d), username=%s",
                (long) sechook_ctx->hook_ctx,
```

```
    sechook_ctx->username) ;  
break;
```

The following code is used to register the function to Attunity Connect:

```
/* Registration function */  
EXPORT_SYMBOL void sechook_register(GDB_HELP_FUNCTIONS  
*help)  
{  
    gdb_help = help;  
  
    GDB_SET_SECHOOK(SecHook);  
}
```

The addon.def file shown above (see "Setting Up Security Hooks" on page 78) registers this function with Attunity Connect. When Attunity Connect is initialized, this function is run as a startup function.



# Chapter 5 Defining a Custom ODBC Driver

The Developer SDK enables you to add a new ODBC driver to the Attunity Connect environment. The ODBC driver that you add can be custom-written or supplied by a third-party vendor, provided it conforms to the ODBC API described in this section.

This section describes the following topics:

- Specifying ODBC Functions
- Registering a Custom ODBC Driver to Attunity Connect
- ODBC Functions

## Specifying ODBC Functions

The generic ODBC API is described in the `odbcfn.h` header file which is part of the Developer SDK installation (see page 19). The functions listed in this file are standard ODBC functions.

When you add an ODBC driver definition (either custom-written or from a third-party vendor) to the Attunity Connect environment, make sure that the DLL contains all of the necessary functions.

- ❖ If a requested function is not found in the DLL, Attunity Connect uses the corresponding ODBC API implemented by the ODBC driver manager on the given platform. For details see "ODBC Support" in the *Attunity Connect Reference*.

## Registering a Custom ODBC Driver to Attunity Connect

To register an ODBC driver in the Attunity Connect environment, the `addon.def` file must include a section defining the data driver, as described in "Registering Drivers and Adapters to Attunity Connect" on page 25. Attunity Connect registers the custom ODBC driver type when the `addon.def` file is processed, but does not load the driver until a data source of that type is accessed.

The syntax for an ODBC driver "add-on" definition is:

```
[name]  
TYPE=ODBC
```

```
SHAREABLE-NAME=image
INIT-FUNCTION=function
```

where:

**name** – A unique identifier for the ODBC driver. This value is used as the data source type in the <datasource> section of the Attunity Connect binding configuration when you identify a data source to be accessed by this custom ODBC driver. Therefore, the name of an ODBC driver must be unique among both the entries in the addon.def file and the data source types associated with supplied Attunity Connect data drivers.

**image** – The path of the shareable image file (DLL). When binding to the first data source TDP of a registered custom ODBC driver, Attunity Connect attempts to locate the DLL using platform-specific conventions. If Attunity Connect cannot locate the DLL (for example, a specified environment variable or its translated filename does not exist), Attunity Connect writes an error message in the Attunity Connect log file. See "System Notes" on page 253 for additional details about this entry.

- ❖ In this document, "DLL" is a generic term to refer to a shareable image or shared library of functions, as appropriate to the specific platform where Attunity Connect runs.

**function** – The name of a function in the DLL that fills in the ODBC function pointer structure (ODBC\_FUNCTIONS). The INIT-FUNCTION statement is optional:

- If the INIT-FUNCTION is specified, Attunity Connect searches the DLL for this entry point and calls the function with two arguments: a pointer to the ODBC\_FUNCTIONS structure and the address of the predefined support functions structure (GDB\_HELP\_FUNCTIONS).
  - ❖ The DLL must expose the specified INIT-FUNCTION for Attunity Connect to reference the functions in a custom ODBC driver. The symbol specified for the INIT-FUNCTION cannot be static. Different platforms provide different mechanisms for exposing a function in a DLL; consult the system documentation for details.
- If the INIT-FUNCTION statement is omitted, Attunity Connect searches the DLL for each of the ODBC functions.
  - ❖ Do not specify INIT-FUNCTION when adding a third-party ODBC driver to the Attunity Connect environment. There are currently no such drivers that support an interface to the function pointer structures that Attunity Connect defines in the Developer SDK.

## ODBC Functions

### SQLAllocConnect

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLALLOCCONNECT) (
    SQLHENV henv,
    SQLHDBC*phdbc);
```

### SQLAllocEnv

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLALLOCENV) (
    SQLHENV*phenv);
```

### SQLAllocStmt

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLALLOCSTMT) (
    SQLHDBC hdbc,
    SQLHSTMT*phstmt);
```

### SQLBindCol

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLBINDCOL) (
    SQLHSTMT hstmt,
    SQLSMALLINT icol,
    SQLSMALLINT iType,
    SQLPOINTER rgbValue,
    SQLINTEGER cbValueMax,
    SQLINTEGER* pcbValue);
```

### SQLBindParameter

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLBINDPARAMETER) (
    SQLHSTMT hstmt,
    SQLSMALLINT ipar,
    SQLSMALLINT fParamType,
    SQLSMALLINT fCType,
    SQLSMALLINT fSqlType,
    SQLINTEGER cbColDef,
    SQLSMALLINT ibScale,
    SQLPOINTER rgbValue,
    SQLINTEGER cbValueMax,
    SQLINTEGER* pcbValue);
```

## SQLColumns

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLCOLUMNS) (  
    SQLHSTMThstmt,  
    SQLCHAR*szTableQualifier,  
    SQLSMALLINTcbTableQualifier,  
    SQLCHAR*szTableOwner,  
    SQLSMALLINTcbTableOwner,  
    SQLCHAR*szTableName,  
    SQLSMALLINTcbTableName,  
    SQLCHAR*szColumnName,  
    SQLSMALLINTcbColumnName);
```

## SQLConnect

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLCONNECT) (   
    SQLHDBCChdbc,  
    SQLCHAR*szDSN,  
    SQLSMALLINTcbDSN,  
    SQLCHAR*szUID,  
    SQLSMALLINTcbUID,  
    SQLCHAR*szAuthStr,  
    SQLSMALLINTcbAuthStr);
```

## SQLDescribeCol

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLDESCRIBECOL) (   
    SQLHSTMThstmt,  
    SQLSMALLINTicol,  
    SQLCHAR*szColName,  
    SQLSMALLINTcbColNameMax,  
    SQLSMALLINT*pcbColName,  
    SQLSMALLINT*pfSqlType,  
    SQLINTEGER*pcbColDef,  
    SQLSMALLINT*pibScale,  
    SQLSMALLINT*pfNullable);
```

## SQLDescribeParam

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLDESCRIBEPARAM) (   
    SQLHSTMThstmt,  
    SQLSMALLINTipar,  
    SQLSMALLINT*pfSqlType,  
    SQLINTEGER*pcbColDef,  
    SQLSMALLINT*pibScale,  
    SQLSMALLINT*pfNullable);
```

## SQLDisconnect

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLDISCONNECT) ( SQLHDBCChdbc );

## SQLDriverConnect

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLDRIVERCONNECT) ( SQLHDBCChdbc, SQLHWNDhwnd, SQLCHAR\*szConnStrIn, SQLSMALLINTcbConnStrIn, SQLCHAR\*szConnStrOut, SQLSMALLINTcbConnStrOutMax, SQLSMALLINT\*pcbConnStrOut, SQLUSMALLINT fDriverCompletion);

## SQLError

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLERROR) ( SQLHENVhenv, SQLHDBCChdbc, SQLHSTMTHstmt, SQLCHAR\*szSqlState, SQLINTEGER\*pfNativeError, SQLCHAR\*szErrorMsg, SQLSMALLINTcbErrorMsgMax, SQLSMALLINT\*pcbErrorMsg);

## SQLExecDirect

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLEXECDIRECT) ( SQLHSTMTHstmt, SQLCHAR\*szSqlStr, SQLINTEGERcbSqlStr);

## SQLExecute

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLEXECUTE) ( SQLHSTMTHstmt);

## SQLExtendedFetch

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLEXTENDEDFETCH) ( SQLHSTMTHstmt, SQLSMALLINTfOrient,

```
SQLINTEGER RfOffset,
SQLINTEGER *pcrow,
SQLSMALLINT *rgfRowStatus);
```

## SQLFetch

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLFETCH) (SQLHSTMT hstmt);

## SQLForeignKeys

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLFOREIGNKEYS) (SQLHSTMT hstmt,  
SQLCHAR \*szPkTableQualifier,  
SQLSMALLINT cbPkTableQualifier,  
SQLCHAR \*szPkTableOwner,  
SQLSMALLINT cbPkTableOwner,  
SQLCHAR \*szPkTableName,  
SQLSMALLINT cbPkTableName,  
SQLCHAR \*szFkTableQualifier,  
SQLSMALLINT cbFkTableQualifier,  
SQLCHAR \*szFkTableOwner,  
SQLSMALLINT cbFkTableOwner,  
SQLCHAR \*szFkTableName,  
SQLSMALLINT cbFkTableName);

## SQLFreeConnect

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLFREECONNECT) (SQLHDBC hdbc);

## SQLFreeEnv

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLFREEENV) (SQLHENV henv);

## SQLFreeStmt

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLFREEESTMT) (SQLHSTMT hstmt,  
SQLSMALLINT fOption);

## SQLGetConnectOption

**Prototype** SQLRETURN (SQL\_API \*ODBC\_API\_SQLGETCONNECTOPTION) (SQLHDBC hdbc,

```
SQLSMALLINTiopt,  
SQLPOINTERoval);
```

## SQLGetData

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLGETDATA) (  
    SQLHSTMT hstmt,  
    SQLSMALLINT icol,  
    SQLSMALLINT fCType,  
    SQLPOINTER rgbValue,  
    SQLINTEGER cbValueMax,  
    SQLINTEGER *pcbValue);
```

## SQLGetFunctions

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLGETFUNCTIONS) (  
    SQLHDBC hdbc,  
    SQLSMALLINT fFunction,  
    SQLSMALLINT *pfExists);
```

## SQLGetInfo

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLGETINFO) (  
    SQLHDBC hdbc,  
    SQLSMALLINT fInfoType,  
    SQLPOINTER rgbInfoValue,  
    SQLSMALLINT cbInfoValueMax,  
    SQLSMALLINT *pcbInfoValue);
```

## SQLGetTypeInfo

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLGETTYPEINFO) (  
    SQLHSTMT hstmt,  
    SQLSMALLINT fSqlType);
```

## SQLNumParams

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLNUMPARAMS) (  
    SQLHSTMT hstmt,  
    SQLSMALLINT *pcpar);
```

## SQLNumResultCols

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLNUMRESULTCOLS) (  
    SQLHSTMT hstmt,  
    SQLSMALLINT *pccol);
```

## SQLParamData

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLPARAMDATA) (  
    SQLHSTMThstmt,  
    SQLPOINTER*Value);
```

## SQLPrepare

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLPREPARE) (   
    SQLHSTMThstmt,  
    SQLCHAR*szSqlStr,  
    SQLSMALLINTcbSqlStr);
```

## SQLPrimaryKeys

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLPRIMARYKEYS) (   
    SQLHSTMThstmt,  
    SQLCHAR*szTableQualifier,  
    SQLSMALLINTcbTableQualifier,  
    SQLCHAR*szTableOwner,  
    SQLSMALLINTcbTableOwner,  
    SQLCHAR*szTableName,  
    SQLSMALLINTcbTableName);
```

## SQLProcedureColumns

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLPROCEDURECOLUMNS) (   
    SQLHSTMThstmt,  
    SQLCHAR*szProcQualifier,  
    SQLSMALLINTcbProcQualifier,  
    SQLCHAR*szProcOwner,  
    SQLSMALLINTcbProcOwner,  
    SQLCHAR*szProcName,  
    SQLSMALLINTcbProcName,  
    SQLCHAR*szColumnName,  
    SQLSMALLINTcbColumnName);
```

## SQLProcedures

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLPROCEDURES) (   
    SQLHSTMThstmt,  
    SQLCHAR*szProcQualifier,  
    SQLSMALLINTcbProcQualifier,  
    SQLCHAR*szProcOwner,  
    SQLSMALLINTcbProcOwner,
```

```
SQLCHAR*szProcName,  
SQLSMALLINTcbProcName);
```

## SQLPutData

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLPUTDATA) (  
    SQLHSTMT hstmt,  
    SQLPOINTER Data,  
    SQLINTEGER Len);
```

## SQLRowCount

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLROWCOUNT) (  
    SQLHSTMT hstmt,  
    SQLINTEGER* pcrow);
```

## SQLSetConnectOption

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLSETCONNECTOPTION) (  
    SQLHDBC hdbc,  
    SQLSMALLINT fOption,  
    SQLINTEGER vParam);
```

## SQLSetStmtOption

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLSETSTMTOPTION) (  
    SQLHSTMT hstmt,  
    SQLSMALLINT fOption,  
    SQLINTEGER vParam);
```

## SQLStatistics

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLSTATISTICS) (  
    SQLHSTMT hstmt,  
    SQLCHAR* szTableQualifier,  
    SQLSMALLINT cbTableQualifier,  
    SQLCHAR* szTableOwner,  
    SQLSMALLINT cbTableOwner,  
    SQLCHAR* szTableName,  
    SQLSMALLINT cbTableName,  
    SQLSMALLINT fUnique,  
    SQLSMALLINT fres);
```

## SQLTables

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLTABLES) (  
    SQLHSTMThstmt,  
    SQLCHAR*szTableQualifier,  
    SQLSMALLINTcbTableQualifier,  
    SQLCHAR*szTableOwner,  
    SQLSMALLINTcbTableOwner,  
    SQLCHAR*szTableName,  
    SQLSMALLINTcbTableName,  
    SQLCHAR*szTableType,  
    SQLSMALLINTcbTableType);
```

## SQLTransact

```
Prototype SQLRETURN (SQL_API *ODBC_API_SQLTRANSACT) (   
    SQLHENVhenv,  
    SQLHDBCChdbc,  
    SQLSMALLINTfType);
```

# **Part III**

# **Application Adapters**



# Chapter 6    The Application Adapter XML Definition Document

An application adapter is built as a shared library (a DLL on Windows) that is loaded by Attunity Connect when a client requires interaction with the application mapped by the adapter.

The adapter code implements various events such as Connect, Disconnect, Execute, TransactionCommit, etc. The adapter needs to implement only events that make sense in its domain. For example, an adapter that maps an application with no transaction semantics would not have to implement any transaction related events.

In addition to the standard events, the adapter code implements the code for the various interactions with the underlying application. Such interactions are either handled using the Execute event or by setting up a special event handler for each particular interaction.

Attunity Connect uses an XML definition document to describe the adapter, its interactions and their input and output structures. NAV\_UTIL PROTOGEN – a utility for use with the adapters SDK – generates C structures and an in-memory compiled representation of the adapter schema for use in run-time. The Attunity Connect mapping engine uses this schema to convert incoming XML requests into the structures required by the adapter, and then convert structures used by the adapter back into XML.

## Application Adapter Definition File

The application adapter definition describes an application adapter, providing the Attunity Connect with all the information it needs to use the application adapter.

The adapter definition can be generated automatically from a COBOL copybook, using the COB\_ACX utility. For details of this utility and how to simplify a schema generated by the utility, see *Attunity Connect Reference*.

## Application Adapter Definition File

A definition contains the following information:

- **General Adapter Properties**

This part defines various simple adapter properties such as name, type, description, time-out values, etc.

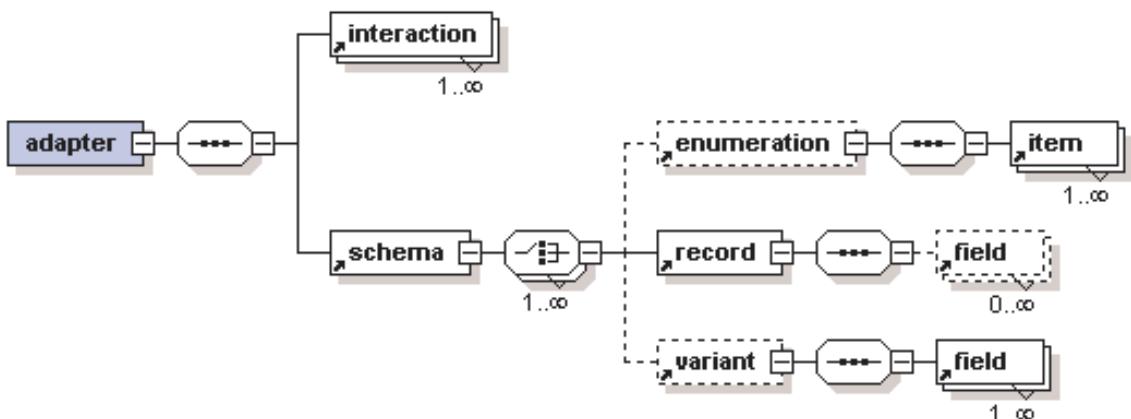
- **Interactions List**

This part lists the interactions offered by the adapter. Information items include the interaction name, its description and input and output record names.

- **Input and Output Records Schema**

This part details the structure of all input and output records used by the adapter.

The following diagram shows the schema of the adapter XML definition document.



### Calc Application Adapter XML

The following XML code is the complete calc application adapter definition, used in this topic to describe the XML syntax.

```

<?xml version="1.0" ?>
<adapter name="calc"
         description="Attunity Connect Calc Schema"
         version="1.0" vendor="Attunity Ltd."
         transactionLevelSupport="OPC"
         authenticationMechanism="basic-password"
         maxActiveConnections="0" maxIdleTimeout="600"
         maxRequestSize="32000">
  <!-- ++++++-->
  <!-- List   a l l   I n t e r a c t i o n s -->

```

## Application Adapter Definition File

```
<!-- ++++++-->
<!-- Simple Arithmetic only -->
<!-- ++++++-->
<!-- Some Binary Arithmetics-->
<!-- ..... -->
<interaction name="add" description="Add 2 numbers"
             mode="sync-send-receive" input="binput"
             output="output" />
<interaction name="sub" description="Subtract 2 numbers"
             mode="sync-send-receive" input="binput"
             output="output" />
<interaction name="mul" description="Multiply 2 numbers"
             mode="sync-send-receive" input="binput"
             output="output" />
<interaction name="div" description="Divide a number by another"
             mode="sync-send-receive" input="binput"
             output="output" />
<!-- Display -->
<!-- ++++++-->
<interaction name="display"
             description="Display a message in the output stream"
             mode="sync-send-receive" input="inpmsg"
             output="outmsg" />
<!-- ..... -->
<!-- The Application Schema -->
<!-- ..... -->
<schema name="calc" version="1.0" header="calcdefs.h">
    <record name="binput">
        <field name="p1" type="int" />
        <field name="p2" type="int" />
    </record>
    <!-- Used for Binary arithmetics input -->
    <!-- ..... -->
    <record name="output">
        <field name="result" type="int" />
    </record>
    <!-- General purpose numeric output -->
    <!-- ..... -->
    <record name="inpmsg">
        <field name="m" type="string" />
    </record>
    <record name="outmsg">
        <field name="m" type="string" nativeType="string" length="512" />
    </record>
</schema>
</adapter>
```

## Application Adapter Definition File

### The adapter Element

The adapter element is the root element of the adapter definition XML document.

- The attributes of the adapter element define simple adapter properties (see below).
- The interaction elements under the adapter describe particular interactions.
- The schema element under the adapter provides the schema of all the records used within the adapter.

The following table summarizes the attributes of the adapter element.

Attribute	Type	Default	Description
authenticationMechanism	enum	basic-password	<p>The type of authentication implemented by the adapter, as follows:</p> <p><b>none</b> – The adapter does not handle authentication.</p> <p><b>basic-password</b> – The adapter implements basic username-password authentication.</p> <p>❖ Kerberos authentication will be supported in future releases.</p>
connect	string		Adapter-specific connect string.
connectionPoolingSize	int		The number of connections that can be held in the connections pool simultaneously.
description	string		A description of the application adapter.
maxActiveConnections	int		The maximum number of simultaneous connections an adapter may take (per process).
maxIdleTimeout	int	600	The maximum time, in seconds, that an active connection can stay idle. After that time, the connection is soft-closed and placed in the connections pool or simply destroyed (depending on the pooling settings).
maxRequestSize	int		The maximum size in bytes that an XML ACX request or reply may span. Larger messages are rejected with an error.

## Application Adapter Definition File

Attribute	Type	Default	Description
name	string		The name of the adapter definition. (This name is normally the name of the adapter specified in the binding configuration. If this name differs from the name in the binding configuration, the binding entry must include a definition element set to the name specified here.)
operatingSystem	string		The operating system the application adapter runs under.
poolingTimeout	int	120	The maximum amount of time (in seconds) that a connection is kept in the connections pool before it is destroyed.
schemaName	string		The name of the schema used to define the adapter.
transactionLevelSupport	enum	1PC	The level of transaction support, as follows: 0PC – no transaction support 1PC – simple (single phase) transactions 2PC – distributed (two phase) transactions
type	string		The name of the adapter executable.
vendor	string		Informational field that comes from the adapter itself.
version	double		Adapter version number.

**Example**

```
<adapter name="calc" description="Attunity Connect Calc Schema"
version="1.0" vendor="Attunity Ltd."
transactionLevelSupport="0PC"
authenticationMechanism="basic-password"
maxActiveConnections="0" maxIdleTimeout="600"
maxRequestSize="32000">
```

## The interaction Element

The interaction element describes a single adapter interaction. The interaction element is a child-element of the adapter element. The following table summarizes the attributes of the interaction element.

Attribute	Type	Default	Description
description	string		A description of the interaction.
input	string		The name of the input record structure.
mode	enum	sync-send-receive	The interaction mode: <b>sync-send-receive</b> – The interaction sends a request and expects to receive a response. <b>sync-send</b> – The interaction sends a request and does not expect to receive a response. <b>sync-receive</b> – The interaction expects to receive a response. <b>async-send</b> – The interaction sends a request that is divorced from the current interaction. This mode is used with events, to identify an event request.
name	string		The name of the interaction.
output	string		The name of the output record structure.

### Example

```
<interaction name="add" description="Add 2 numbers"
            mode="sync-send-receive" input="binput" output="output" />
<interaction name="display"
            description="Display message in output stream"
            mode="sync-send-receive" input="inpmsg" output="outmsg" />
```

## The schema Element

The schema element describes the structures used in the interactions. The schema element is a child-element of the adapter element. Only one schema is allowed per adapter. The following table summarizes the attributes of the schema element.

Attribute	Type	Description
header	string	The include file generated by NAV_UTIL PROTOGEN.

Attribute	Type	Description
initialization header	string	???????
name	string	The name of the adapter.
noAlignment	boolean	Determines whether buffers are aligned or not.
version	string	The schema version.

### Example

```
<schema name="calc" version="1.0" header="calcdefs.h">
```

## The enumeration Element

The enumeration element defines an enumeration type for use in interaction definitions.

## The record Element

The record element defines a grouping of fields. The following table summarizes the attributes of the record element.

Attribute	Type	Description
EntryRef	string	(Used with the COM adapter.) The name of the method to be invoked within that object.
IID	string	(Used with the COM adapter.) The UUID of a user defined type. Used for user defined data types only.
libIID	string	(Used with the COM adapter.) The UUID of the library in which the user defined data type is defined. Used for user defined data types only.
noAlignment	boolean	Determines whether buffers are aligned or not.
name	string	The name of the record.
ObjectRef	string	(Used with the COM adapter.) Either a ProgID or a UUID of the COM that this input record refers to.
ParamCount	int	(Used with the COM adapter.) The number of parameters passed to that method.

---

## Application Adapter Definition File

Attribute	Type	Description
program (OS/390 and z/OS only)	string	(Used with the CICS adapter.) The name of the program to be executed in a CICS transaction.
transid (OS/390 and z/OS only)	string	(Used with the CICS adapter.) The CICS transaction id where the program will run. The TRANSID must be EXCI or a copy of this transaction.

### Example

```
<record name="binput">
  <field name="p1" type="int" />
  <field name="p2" type="int" />
</record>
<record name="output">
  <field name="result" type="int" />
</record>
<record name="inpmsg">
  <field name="m" type="string" />
</record>
<record name="outmsg">
  <field name="m" type="string" nativeType="string" length="512" />
</record>
```

### Defining Hierarchies

The variant record element can be used to define a hierarchical structure. The hierarchical definition includes a record definition for the child. The parent record includes a field record with the type name that is used to define the child.

### Example

```
<record name="parent">
  <field name="f1" type="child" />
  <field name="f2" type="int" />
</record>
<record name="child">
  <field name="c1" type="int" />
  <field name="c2" type="string" />
  <field name="c3" type="string" />
</record>
```

The XML used to access the adapter, must use the same structure as specified in the interaction definition.

## The variant record Element

The variant record element defines a variant structure. The variant definition includes a list of record definitions that define the variant fields. The following table summarizes the attributes of the variant element.

Attribute	Type	Description
autoSelect	boolean	Determines the case (upper or lower). This attribute is for internal use only.
implicit	true	For internal use only.
name	string	The name of the variant structure.

### Example

```
<record name="v1">
  <field name="f1" type="string" nativeType="string" size="40" />
</record>
<record name="v2">
  <field name="f1" type="int" />
</record>
<record name="v3">
  <field name="f1" type="double" />
</record>
<variant name="unionRec" autoSelect="true" implicit="true">
  <field name="var1" type="v1" />
  <field name="var2" type="v2" />
  <field name="var3" type="v3" />
</variant>
```

## The field Element

The field element defines single data item within a record or a variant. The following table summarizes the attributes of the variant element.

Attribute	Type	Description
array	array	An array field that is made up of other fields.
COMtype	enum	(Used with the COM adapter.) Specifies the field's data type as recognized by COM, using explicit COM enumeration values (for details, see "COM Application Adapter").

## Application Adapter Definition File

Attribute	Type	Description
counter	int	Runtime value holding the actual number occurrences in the array. A field can be specified as a counter field.
default	string	The default value for the field. The default value for an integer is zero (0) and for a string NULL. Specifying a default value means that the field can be omitted from the input XML. ❖ If a field isn't nullable, when using the database adapter and a default value is not supplied, an error occurs.
filter		Filtering of extraneous, unwanted metadata. This attribute is for internal use only.
length	int	The size of the field including a null terminator, when the data type supports null termination (such as the cstring data type).
mechanism	string	(Used with the LegacyPlug adapter.) The method by which the field is passed or received by the procedure (either byValue or byReference).  When a parameter is used for both input and output, the mechanism must be the same for both the input and the output.  For outer-level (non-nested) parameters, structure parameters (for the structure itself, and not structure members), and variant parameters, the default value is byReference.
name	string	The name of the field.
nativeType	string	The Attunity Connect data type for the field. Refer to "ADD Supported Data Types" on page 38 for a list of all supported native data types.
offset	int	An absolute offset for the field in a record.
paramnum	int	(Used with the LegacyPlug adapter.) The procedure argument number. 0 indicates the value is a return value. 1 indicates the value is the first argument, and so on.  If paramNum is specified at the record level, it cannot be specified for any of the record members (at the field level).
precision	int	The float data type precision. Used in conjunction with scale (see below).
private	boolean	The value is hidden in the response.
reference	boolean	Used with array (see above), to identify a pointer.
required	boolean	A value is mandatory for this field.

Attribute	Type	Description
scale	int	The float data type scale. Used in conjunction with precision (see above).
size	int	The size of the field.
type	string	The data type of the field. The following are valid data types: <ul style="list-style-type: none"> <li>■ Binary</li> <li>■ Boolean</li> <li>■ Byte</li> <li>■ Date</li> <li>■ Double</li> <li>■ Enum</li> <li>■ Float</li> <li>■ Int</li> <li>■ Long</li> <li>■ Numeric[(p,s)]</li> <li>■ Short</li> <li>■ String</li> <li>■ Time</li> <li>■ Timestamp</li> </ul>
Usage	string	(Used with the COM adapter.) Explains what the COM adapter is about to do with this field: <b>InstanceTag</b> – Names an object instance. <b>Property</b> – Treated as a property. <b>Parameter</b> – The field value should be passed as a parameter to/from a method. <b>RetVal</b> – The field will hold a method's return value.
value	string	Internal code representing the field, used in the C program.

### Example

```
<field name="m" type="string" nativeType="string"
      length="512" private="true" />
```

## The Type Schema

A type schema enables an adapter to expose information about required configuration parameters.

## Application Adapter Definition File

The schema is specified as a separate schema in an XML file with the following format:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<schema name="name" version="1.0" description="description"
    header="header.h" initializationHeader="iheader.c">
<!-- -->
<record name="config">
    <field name="fname" type="datatype" />
    ...
</record>
</schema>
```

The record name is always "config". The table below summarizes the attributes of the record element.

Attribute	Type	Description
name	string	The name of the schema.
fname	string	The name of the field used in the <config> statement in the binding configuration. The value of fname cannot be "action"

To use the type schema, use NAV\_UTIL PROTOGENN to generate the header (*header.h*) and the initialization header *iheader.c*.

Include the schema in the adapter initialization function (see "The INITIALIZE Function" on page 112).

### Example

The following type schema is used with the sample ordersys adapter (supplied with the Attunity Connect installation):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema name="ordersysType" version="1.0"
    description="Order system configuration schema"
    header="ordcfg.h" initializationHeader="ordcfgi.c">
<!-- -->
<record name="config">
    <field name="systemId" type="string" />
    <field name="maxOrders" type="int" />
</record>
</schema>
```

The following entry defines an adapter in the binding configuration for the ordersys adapter:

## Application Adapter Definition File

```
<adapter name='orders' type='ordersys'>
  <config maxOrders='30' systemId='AXZ12784' />
</adapter>
```

Application Adapter Definition File

# **Chapter 7    The Application Adapter API (GAP)**

The Application Adapter Framework (AAF) provides components enabling the creation of adapters to existing applications. The Generic Application Adapter API (GAP) defines the interfaces for adding new application adapters to Attunity Connect. The adapters framework is designed to enable application integration by means of XML and the J2EE Java Connector Architecture (JCA) resource manager provided with Attunity Connect.

The following figure depicts a skeleton adapter. The skeleton is described in detail after the figure.

```

00001:
00002: #include <stdio.h> /* Standard headers */
00003: #include "gap.h" /* GAP headers */
00004:
00005: GDB_HELP_DEFINE; /* Define gdb_help utility context for global use */
00006:
00007: #include "MYAPinit.h" /* Generated from MYAP.XML */
00008:
00009: GAP_STATUS onAction1(GAP_REQUEST *pGapRequest,
00010:                         ACT1_OUT *pOut, ACT1_IN *pIn)
00011: {
00012:     pOut->iResult = some_function_of(pIn); /* Actual interaction work */
00013:     return GAP_STATUS_OK;
00014: }
00015:
00016: GAP_STATUS onConnect(GAP_REQUEST *pGapRequest,
00017:                         ACX_CONNECT_RESPONSE * pConnectResponse,
00018:                         ACX_CONNECT * pConnect)
00019: {
00020:     /* ...code to handle connection... */
00021:     return GAP_STATUS_OK;
00022: }
00023:
00024: static GAP_STATUS onBind(GAP_REQUEST *pGapRequest,
00025:                           void *pDummy, GAP_BIND *pBind)
00026: {
00027:     /* Attach onAction1 handler (a sample adapter interaction) */
00028:     GAP_SET_INTERACTION_INFO(pGapRequest, "action1", onAction1);
00029:     return GAP_STATUS_OK;
00030: }
00031:
00032: GAP_EXPORT register_MYAP_adapter(GAP_HANDLERS *pHandlers,
00033:                                     GDB_HELP_FUNCTIONS *pHelp)
00034: {
00035:     GAP_INITIALIZE(pHelp); /* Set global GAP utility context */
00036:
00037:     pHandlers->onBind = onBind; /* Attach onBind handler */
00038:     pHandlers->onConnect= onConnect; /* Attach onConnect handler */
00039:     pHandlers->pAdapter =
00040:         nv_init_MYAP_adapter(); /* Generated from MYAP.XML */
00041: }
```

Figure 1: An adapter skeleton

**Line 2 – Include of standard headers.**

**Line 3** – Include of the ‘gap.h’ header – the GAP API declarations (this includes the ‘dbgdb.h’ header which defines most of the SDK utility services).

**Line 5** – Definition of ‘gdb\_help’ – the SDK utility context. This context is the mechanism by which the SDK utility services are exposed to the developer. Most of the utility services make use of this context. If this context is only used in one module, this definition could be prefixed with ‘static’ so that ‘gdb\_help’ would be local to this module.

**Line 7** – The adapter must have been described in a special XML adapter definition document. The NAV\_UTIL PROTOGEN utility generates two headers files (whose names can be determined by the developer) that define the adapter specific structures and initialize an in-memory representation of the adapter. Here the initialization header is included.

**Line 9** – This is an example of a typical adapter specific interaction handler. All interaction handlers are similar: The first parameter is the request context, then the output structure and last is the input structure (as described in the XML definition). This interaction handler gets called upon dispatching an Execute request for an interaction named “action1”. The interaction handler is registered in line 28.

**Line 12** – The interaction code is supposed to take whatever information it needs from the input structure (which represents the contents of the incoming Execute request), processes it and updates the output structure with the desired results. The output structure is translated to XML and returned to the caller.

**Line 13** – A typical successful interaction returns GAP\_STATUS\_OK.

**Line 16** – The OnConnect function handles the Connect event. It implements Connect with the underlying application. Adapter name and authorization information are provided in the input structure (pConnect).

**Line 24** – The OnBind function handles the Bind event which is a local event during which the adapter can register handlers for particular interactions. In this example, the onAction1() function is associated with the ‘action1’ interaction. The GAP\_SET\_INTERACTION\_INFO() function is one of the GAP specific services provided by the SDK.

**Line 32** – The adapter registration function. This function must be exported from the shared library (hence the GAP\_EXPORT prefix) so that it can be dynamically loaded by Attunity Connect. This function gets as parameters a handlers registration structure as well as the SDK utility context.

**Line 35** – The global utility context is initialized with the context passed as a parameter to the registration function.

**Line 37** – Event handlers are registered on the handlers registration structure. Events for which no handlers are registered are automatically handled by the framework.

**Line 39** – The nv\_init\_MYAP\_adapter() function was generated by the NAV\_UTIL PROTOGEN utility, based on the adapter's XML definition document. Calling it returns an in-memory representation of the original XML definition document. This in-memory definition must be stored as shown for the adapter to work.

#### The Application Adapter API Functions

An application adapter definition includes routines from some or all of the following categories:

- Initialization function – see below.
- Connection Related functions – see page 113.
- Transaction related functions – see page 116.
- The Execute function – see page 119.
- Metadata Related functions – see page 120.
- The Ping function – see page 121.
- The Bind function – see page 121.
- The exception handling functions – see page 122.
- The event handling functions – see page 124.
- ❖ The syntax uses C terminology, followed by the equivalent COBOL function name.

## The INITIALIZE Function

#### Description

The adapter registration function. This function is required. This function must be exported (exposed) by the DLL. Attunity Connect searches the DLL for the initialization function using the name given in the INIT-FUNCTION assignment of the addon.def file.

#### Prototype

```
GDB_EXPORT (*GAP_INITIALIZE) (
    GAP_HANDLERS *handlers,
    GAP_HELP_FUNCTIONS *help);
```

Parameter	Usage	Description
handlers	Output	A pointer to structure of handlers/triggers which AAF will call.

Parameter	Usage	Description
help	Input	A pointer to a structure (GAP_HELP_FUNCTIONS) that defines various predefined support functions supplied by Attunity Connect for use with the Developer SDK.

## Connection APIs

The following functions handle the connection and connection context for a request:

- The CONNECT Function
- The SET\_CONNECTION Function
- The DISCONNECT Function
- The REAUTHENTICATE Function
- The CLEAN\_CONNECTION Function
- The HOLD\_CONNECTION Function

### The CONNECT Function

**Description** The ‘onConnect’ event is called upon an ACX ‘connect’ request. The adapter should check the authentication information and create its own connection structure and store it in pGapRequest->pConnection.

**Prototype**

```
GAP_STATUS (*GAP_ON_CONNECT) (
    GAP_REQUEST *GapRequest,
    ACX_CONNECT_RESPONSE *ConnectResponse,
    ACX_CONNECT *Connect);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
ConnectResponse	Output	A connection id and the timeout that will be used with this connection.
Connect	Input	The connection details, including the name of the adapter, the timeout that will be used with this connection, and authentication information (a username and password).

### The SET\_CONNECTION Function

**Description** The ‘onSetConnection’ event is called upon an ACX ‘setConnection’ request. The adapter does not necessarily have to do anything here. The

event means that work is resumed on this connection (typically after ‘holdConnection’ or ‘cleanConnection’).

**Prototype**

```
GAP_STATUS (*GAP_ON_SET_CONNECTION) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    ACX_SET_CONNECTION *SetConnection);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
SetConnection	Input	The connection id returned as part of the ConnectResponse by the CONNECT function.

## The DISCONNECT Function

**Description**

The ‘onDisconnect’ event is called upon an ACX ‘disconnect’ request or when Attunity Connect closes a connection (for example, if connection idle time-out expires).

If a transaction is manually started, auto commit is disabled for the duration of the transaction.

If auto commit is disabled and ‘execute’ occurs, then a (local) transaction is automatically started.

**Prototype**

```
GAP_STATUS (*GAP_ON_DISCONNECT) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    void *DummyIn);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
DummyIn	Input	A dummy input placeholder.

## The REAUTHENTICATE Function

**Description**

The ‘onReauthenticate’ event gets called upon ACX ‘reauthenticate’ request. The adapter is not required to implement the reauthentication of an established connection.

**Prototype**

```
GAP_STATUS (*GAP_ON_REAUTHENTICATE) (
    GAP_REQUEST *GapRequest,
```

```
void *DummyOut,
ACX_REAUTHENTICATE *Reauthenticate);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
Reauthenticate	Input	The username and password for the new client using the connection.

## The CLEAN\_CONNECTION Function

<b>Description</b>	The ‘onCleanConnection’ event is called upon an ACX ‘cleanConnection’ request. The adapter may take this opportunity to release resources that would no longer be used under this connection (for example, cached cursors, etc.)												
<b>Prototype</b>	<pre>GAP_STATUS (*GAP_ON_CLEAN_CONNECTION) (     GAP_REQUEST *GapRequest,     void *DummyOut,     ACX_CLEAN_CONNECTION *CleanConnection);</pre> <table border="1"> <thead> <tr> <th>Parameter</th><th>Usage</th><th>Description</th></tr> </thead> <tbody> <tr> <td>GapRequest</td><td>Input</td><td>The handle to the request context.</td></tr> <tr> <td>DummyOut</td><td>Output</td><td>A dummy output placeholder.</td></tr> <tr> <td>CleanConnection</td><td>Input</td><td>Authentication details are cleaned.</td></tr> </tbody> </table>	Parameter	Usage	Description	GapRequest	Input	The handle to the request context.	DummyOut	Output	A dummy output placeholder.	CleanConnection	Input	Authentication details are cleaned.
Parameter	Usage	Description											
GapRequest	Input	The handle to the request context.											
DummyOut	Output	A dummy output placeholder.											
CleanConnection	Input	Authentication details are cleaned.											

## The HOLD\_CONNECTION Function

<b>Description</b>	The ‘onHoldConnection’ event is called when work on a connection is suspended. This may happen when switching to another connection (if the current connection is persistent) or when an ACX request completes. This event is useful in the context of distributed transaction management.												
<b>Prototype</b>	<pre>GAP_STATUS (*GAP_ON_HOLD_CONNECTION) (     GAP_REQUEST *GapRequest,     void *DummyOut,     void *DummyIn);</pre> <table border="1"> <thead> <tr> <th>Parameter</th><th>Usage</th><th>Description</th></tr> </thead> <tbody> <tr> <td>GapRequest</td><td>Input</td><td>The handle to the request context.</td></tr> <tr> <td>DummyOut</td><td>Output</td><td>A dummy output placeholder.</td></tr> <tr> <td>DummyIn</td><td>Input</td><td>A dummy input placeholder.</td></tr> </tbody> </table>	Parameter	Usage	Description	GapRequest	Input	The handle to the request context.	DummyOut	Output	A dummy output placeholder.	DummyIn	Input	A dummy input placeholder.
Parameter	Usage	Description											
GapRequest	Input	The handle to the request context.											
DummyOut	Output	A dummy output placeholder.											
DummyIn	Input	A dummy input placeholder.											

## Transaction APIs

The following functions are available to handle transaction operations:

- The SET\_AUTO\_COMMIT Function
- The TRANSACTION\_START Function
- The TRANSACTION\_END Function
- The TRANSACTION\_COMMIT Function
- The TRANSACTION\_PREPARE Function
- The TRANSACTION\_ROLLBACK Function
- The TRANSACTION\_FORGET Function
- The TRANSACTION\_RECOVER Function

### The SET\_AUTO\_COMMIT Function

**Description** The ‘onSetAutoCommit’ event is called upon an ACX ‘setAutoCommit’ request. The adapter is notified about the desired transaction commitment policy. By default, it is assumed to be enabled.

If a transaction is manually started, auto commit is disabled for the duration of the transaction.

If auto commit is disabled and ‘execute’ occurs, then a (local) transaction is automatically started (that is, ‘GAP\_ON\_TRANSACTION\_START’ would get called).

**Prototype**

```
GAP_STATUS (*GAP_ON_SET_AUTO_COMMIT) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    ACX_SET_AUTO_COMMIT *SetAutoCommit);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
SetAutoCommit	Input	A flag specifying whether autocommit mode is set.

### The TRANSACTION\_START Function

**Description** The ‘onTransactionStart’ event is called upon an ACX ‘transactionStart’ request or when auto-commit is disabled and ACX ‘execute’ request is made. Another case is when distributed transaction is in progress and work on it is being resumed.

**Prototype**

```
GAP_STATUS (*GAP_ON_TRANSACTION_START) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    ACX_TRANSACTION_START *TransactionStart);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
TransactionStart	Input	The state of the transaction (join, resume or empty).

**The TRANSACTION\_END Function****Description**

The ‘onTransactionEnd’ event is called upon an ACX ‘transactionEnd’ request or when distributed transaction is in progress and work on the transaction is being suspended.

**Prototype**

```
GAP_STATUS (*GAP_ON_TRANSACTION_END) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    ACX_TRANSACTION_END *TransactionEnd);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
TransactionEnd	Input	The state of the transaction: success, suspend or fail.

**The TRANSACTION\_COMMIT Function****Description**

The ‘onTransactionCommit’ event is called upon an ACX ‘transactionCommit’ request. If the adapter supports transactions, this is the point where it should commit the work done under the current transaction.

**Prototype**

```
GAP_STATUS (*GAP_ON_TRANSACTION_COMMIT) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    ACX_TRANSACTION_COMMIT *TransactionCommit);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
TransactionCommit	Input	Specifies whether the adapter uses a one-phase commit protocol.

## The TRANSACTION\_PREPARE Function

**Description** The ‘onTransactionPrepare’ event is called upon an ACX ‘transactionPrepare’ request. If the adapter supports distributed transactions, this is the point where it should prepare to commit the work done under the specified distributed transaction.

This event only gets triggered for distributed transactions.

**Prototype**

```
GAP_STATUS (*GAP_ON_TRANSACTION_PREPARE) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    ACX_TRANSACTION_PREPARE *TransactionPrepare);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
TransactionPrepare	Input	Whether the transaction is prepared or not.

## The TRANSACTION\_ROLLBACK Function

**Description** The ‘onTransactionRollback’ event is called upon an ACX ‘transactionRollback’ request, or when a connection with an ongoing transaction is closed.

If the adapter supports transactions, this is the point where it should roll back the work done under the current transaction.

**Prototype**

```
GAP_STATUS (*GAP_ON_TRANSACTION_ROLLBACK) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    ACX_TRANSACTION_ROLLBACK *TransactionRollback);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
TransactionRollback	Input	Whether the transaction rolls back or not.

## The TRANSACTION\_FORGET Function

**Description** The ‘onTransactionForget’ event is called upon an ACX ‘transactionForget’ request.

This event is triggered only for adapters supporting distributed transactions.

**Prototype**

```
GAP_STATUS (*GAP_ON_TRANSACTION_FORGET) (
    GAP_REQUEST *GapRequest,
    void *DummyOut,
    ACX_TRANSACTION_FORGET *TransactionForget);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
TransactionForget	Input	Delete the completed transaction branch.

**The TRANSACTION\_RECOVER Function****Description**

The ‘onTransactionRecover’ event is called upon an ACX ‘transactionRecover’ request.

This event only gets triggered for adapters supporting distributed transactions.

**Prototype**

```
GAP_STATUS (*GAP_ON_TRANSACTION_RECOVER) (
    GAP_REQUEST *GapRequest,
    ACX_TRANSACTION_RECOVER_RESPONSE
        *TransactionRecoverResponse,
    ACX_TRANSACTION_RECOVER *TransactionRecover);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
TransactionRecoverResponse	Output	The number of items returned.
TransactionRecover	Input	The maximum number of items to return.

**The EXECUTE Function****Description**

The ‘onExecute’ event is called upon an ACX ‘execute’ request. Here the actual adapter interactions should be performed.

**Prototype**

```
GAP_STATUS (*GAP_ON_EXECUTE) (
    GAP_REQUEST *GapRequest,
```

```
ACX_EXECUTE_RESPONSE *ExecuteResponse,  
ACX_EXECUTE *Execute);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
ExecuteResponse	Output	The output record returned by the function.
Execute	Input	The input record to be executed.

## Metadata APIs

The following functions are available to handle metadata:

- The GET\_METADATA\_ITEM Function
- The GET\_METADATA\_LIST Function

### The GET\_METADATA\_ITEM Function

**Description** The 'onGetMetadataItem' event is called upon an ACX 'getMetadataItem' request. Handling of metadata requests for the adapter's basic features (such as interactions, adapter and records schema) is handled automatically

**Prototype**

```
GAP_STATUS (*GAP_ON_GET_METADATA_ITEM) (  
    GAP_REQUEST *GapRequest,  
    ACX_GET_METADATA_ITEM_RESPONSE  
        *GetMetadataItemResponse,  
    ACX_GET_METADATA_ITEM *GetMetadataItem);
```

Parameter	Usage	Description
GapRequest	Input	The handle for the data driver
GetMetadataItemResponse	Output	The output returned by the function.
GetMetadataItem	Input	The item type and name. The following are valid items: <ul style="list-style-type: none"><li>■ adapter</li><li>■ interaction</li><li>■ record</li><li>■ schema</li><li>■ all</li><li>■ event</li><li>■ schema (w3c)</li></ul>

## The GET\_METADATA\_LIST Function

**Description** The ‘onGetMetadataList’ event is called upon an ACX ‘getMetadataList’ request. Handling of metadata requests for the adapter’s basic features (interactions, adapter, records schema) is handled automatically

**Prototype**

```
GAP_STATUS (*GAP_ON_GET_METADATA_LIST) (
    GAP_REQUEST *GapRequest,
    ACX_GET_METADATA_LIST_RESPONSE
        *GetMetadataListResponse,
    ACX_GET_METADATA_LIST *GetMetadataList);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
GetMetadataListResponse	Output	The output returned by the function.
GetMetadataList	Input	The item type and name, the number of results to return and the starting item.

## The PING Function

**Description** The ‘onPing’ event is called upon an ACX ‘ping’ request.

**Prototype**

```
GAP_STATUS (*GAP_ON_PING) (
    GAP_REQUEST *GapRequest,
    ACX_PING_RESPONSE *AcxPing,
    void *DummyIn);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
AcxPingResponse	Output	The output returned by the function.
DummyIn	Input	A dummy input placeholder.

## The BIND Function

**Description** The ‘onBind’ event is called upon an initial binding of the adapter (no connection is open at that point). In this event, the adapter definition is already loaded and the adapter code may do various process-scope set up. This event is guaranteed to occur in one thread only, and just once (assuming the binding is not failed).

**Prototype**

```
struct GAP_BIND {char *pszConnectionString;} GAP_BIND;
GAP_STATUS (*GAP_ON_BIND) (
```

```
GAP_REQUEST *GapRequest,  
void *DummyOut,  
GAP_BIND *GapBind);
```

Parameter	Usage	Description
GapRequest	Input	The handle to the request context.
DummyOut	Output	A dummy output placeholder.
GapBind	Input	Handle to binding information (as received in the connect string).

## Exception Handling

Attunity Connect provides the following exception handling services:

- Software exceptions within the adapter code which are not caught by the adapter itself are automatically caught by Attunity Connect and are reported to the caller as an ACX Exception verb.
- The adapter itself may catch software exceptions or otherwise detect error conditions and report them back using the GAP\_SET\_EXCEPTION[\_V] service function.

The ACX Exception verb is described below in "The GAP\_SET\_EXCEPTION[\_V] Functions". When exceptions are generated using the GAP API, it is important to adhere to the form and naming conventions with respect to exceptions. Here is a short summary:

- The 'origin' exception field should be of the form <adaptor-type>.<interaction-name>. For example, if the adapter 'db2' which is of type "query" has an exception executing the 'update' interaction, then the appropriate origin value is "query.update".
- The 'name' exception field should be of the form <source>.<code> where <source> is either 'client' when the error is attributed to the client (for example, when the parameters are wrong or when the request refers to a resource which is not known on the server), or it can be 'server' when the error is attributed to the server (for example, when there was an internal server error, or when there is some resource problem on the server). The <code> part is exception specific name (starting with a letter and containing letters and digits). An example of an exception name is "client.xmlError" and "server.internalError".
- The 'info' fields should contain human-readable information about the source of the exception.

## The GAP\_SET\_EXCEPTION[\_V] Functions

<b>Description</b>	The GAP_SET_EXCEPTION() and GAP_SET_EXCEPTION_V() functions allow the adapter to generate an ACX exception to indicate an error in the processing of an ACX request. The _V version gets the error information strings as an array while the other version gets these strings as call parameters.
	Immediately after calling one of these functions, the event handler must exit with a FALSE return code. The ACX dispatcher will stop the processing of the current ACX request and would return the exception to the caller.
<b>Prototype</b>	<pre>void GAP_SET_EXCEPTION(     GAP_REQUEST *GapReq,     char *Origin,     char *Name,     ... , NULL)  void GAP_SET_EXCEPTION_V(     GAP_REQUEST *GapReq,     char *Origin,     char *Name,     int Errors,     char *Errors[])</pre>
<b>Parameters</b>	<p><b>GapReq</b> – A GAP request handle (as passed to the ACX event handler).</p> <p><b>Origin</b> – A string identifying the exception origin. The string is expected to be of the form &lt;adaptor-type&gt;.&lt;interaction-name&gt; (for example, “query.update”).</p> <p><b>Name</b> – A user context to associate with the interaction.</p> <p><b>Errors (int)</b> – Number of error information strings in the Errors array.</p> <p><b>Errors (char *[])</b> – NULL or an array of 1 or more error information string (human readable).</p> <p><b>... (char *)</b> – One or more error information strings. Note that the last one must always be NULL.</p>

## The Adapter Event and Handler Functions

Events are handled by handler functions. The following functions describe the various events and their handler functions.

### The GAP\_SET\_INTERACTION\_INFO Function

<b>Description</b>	<p>When an GAP Execute request is invoked, the onExecute handler is called with an ACX_EXECUTE input parameter which provides the name of the interaction to invoke (InteractionName) along with its input and an interaction definition structure (InteractionDef). The GAP_SET_INTERACTION_INFO() function allows to associate a user structure with a particular interaction definition, so that when an interaction is invoked that structure is available.</p> <p>A typical use of this interaction info option is to store execution information along with the interaction definition (for example, a pointer to the function implementing the interaction).</p> <p>The scope of this association is global – that is, all the threads that use this adapter would see the same interaction info context. Therefore, it is not recommended to change the interaction info context while the adapter is working. The best place to set the interaction information is in the onBind() event which occurs right after the adapter is loaded and before any request on it is processed. Changing the interaction information in any other place requires the adapter code to protect the context from concurrent access.</p> <p>A particular case of the GAP_SET_INTERACTION_INFO() function is that where no onExecute() handler is defined by the adapter. In this case, the pInfo is used to associate an interaction handler (a function pointer) with the interaction. See, for example, line 28 in the sample in Figure 1 where the onAction1() function is associate with the “action1” interaction so that when an “action1” ACX Execute request is dispatched, the onAction1() function is called with the appropriate parameters. Note that this would not work when an onExecute handler is given – in that case, the handler should invoke the interaction function in its own particular way.</p>
<b>Prototype</b>	<pre>void GAP_SET_INTERACTION_INFO(     GAP_REQUEST *GapReq,     char *InteractionName,     void *Info)</pre>

**Parameters** **GapReq** – A GAP request handle (as passed to the ACX event handler).

**InteractionName** – A GAP interaction name with which to associate the user context.

**Info** – A user context to associate with the interaction.

## The GAP\_GET\_INTERACTION\_INFO Function

<b>Description</b>	This function returns the interaction info context that was earlier associated with the given interaction definition. See the discussion of THE GAP_SET_INTERACTION_INFO() function for more details.
<b>Prototype</b>	<pre>void *GAP_GET_INTERACTION_INFO(     ACX_INTERACTION *Interaction)</pre>
<b>Parameters</b>	<b>Interaction</b> – A GAP interaction definition (passed to the onExecute ACX event handler as Execute and InteractionDef).

## The GAP\_GET\_EXECUTE\_INFO Function

<b>Description</b>	This function can be called from an onExecute() handler to get the adapter-specific interaction information associated with the current interaction, as well as the output and input record structures.
<b>Prototype</b>	<pre>void GAP_GET_EXECUTION_INFO(     GAP_REQUEST *GapReq,     ACX_INTERACTION *Interaction,     void *InteractionInfo,     struct XP_RECORD **OutputRecord,     struct XP_RECORD **InputRecord)</pre>
<b>Parameters</b>	<b>GapReq</b> – A GAP request handle (as passed to the ACX event handler). <b>Interaction</b> – A GAP interaction definition (passed to the onExecute ACX event handler as Execute and InteractionDef). <b>InteractionInfo</b> – Adapter specific interaction information. <b>OutputRecord</b> – The structure of the interaction output record. <b>InputRecord</b> – The structure of the interaction input record.

## The GAP\_SET\_ADAPTER\_INFO Function

<b>Description</b>	The GAP_SET_ADAPTER_INFO function associates a user context with the current adapter.
	The scope of this association is global – that is, all the threads that use this adapter would see the same adapter info context. Therefore, the adapter code is required to synchronize access to this context (unless it is set once at the onBind() event and not modified afterwards).
<b>Prototype</b>	<pre>void GAP_SET_ADAPTER_INFO(     GAP_REQUEST *GapReq,     void *Info)</pre>
<b>Parameters</b>	<b>GapReq</b> – A GAP request handle (as passed to the ACX event handler). <b>Info</b> – A user context to associate with the adapter.

## The GAP\_GET\_ADAPTER\_INFO Function

<b>Description</b>	The GAP_SET_ADAPTER_INFO function associates a user context with the current adapter.
	The scope of this association is global – that is, all the threads that use this adapter would see the same adapter info context. Therefore, the adapter code is required to synchronize access to this context (unless it is set once at the onBind() event and not modified afterwards).
<b>Prototype</b>	<pre>void *GAP_GET_ADAPTER_INFO(     GAP_REQUEST *GapReq,     void *Info)</pre>
<b>Parameters</b>	<b>GapReq</b> – A GAP request handle (as passed to the ACX event handler). <b>Info</b> – A user context to associate with the adapter.

## The GAP\_GET\_METADATA\_ATTR Function

<b>Description</b>	This function returns custom attributes of a schema, record or field.
<b>Prototype</b>	<pre>void GAP_GET_METADATA_ATTR (     XP_ATTR *Attrs,     char *AttrName,     char *DefaultValue)</pre>
<b>Parameters</b>	<b>Attrs</b> – The attribute: schema, record or field.

**AttrName** – The name of the attribute.

**DefaultValue** – The default value.



# **Part IV**

# **User Defined Data Types**



# Chapter 8 Defining Data Types

The Attunity Connect Developer SDK includes the User-defined Data Types (UDT) API. You can define additional data types to accommodate the requirements of a data source or to supplement data types available in a standard Attunity Connect driver such as ADD-RMS or ADD-DISAM.

- ❖ All of the data structures and macros described in this chapter are defined in the header files dbgdb.h and dtypeid.h which are part of the Developer SDK installation (see page 19).

This chapter describes the following topics:

- Defining a Data Type
- Registering a User-Defined Data Type to Attunity Connect
- Data Type Functions
- Data Type Support Macros

A column defined with a user-defined data type can be viewed in the Attunity Studio Metadata perspective but not edited (other columns in the table can be edited).

## Defining a Data Type

This section outlines the steps that you must perform to define a data type.

1. As part of the user-defined data type code library, write an external function, which the DLL can expose, that defines the structures and attributes underlying the data type. Include the following logic in this “registration function”:
  - Use the GDB\_INITIALIZE macro to initialize the GDB\_HELP\_FUNCTIONS structure and the version member of the function pointer, to ensure proper use of the support functions. (For details, see page 153.)
  - Populate the UDT\_DATATYPE structure to define the data type. The data type definitions (such as id and size) must be specified explicitly. (For details, see page 290.)
  - Call the macro UDT\_ADD\_DATATYPE, passing a pointer to the UDT\_DATATYPE structure. UDT\_ADD\_DATATYPE registers a user-defined data type to Attunity Connect, using the support

- function DT\_ADD\_DATATYPE (nv\_dt\_add\_datatype). (For details, see page 154.)
2. Include the SDK header files `dbgdb.h` and `dtypeid.h` that are provided as part of the Developer SDK installation (described on page 19).
  3. Establish a pointer to the predefined support functions in `GDB_HELP_FUNCTIONS`, using the macro `GDB_HELP_DEFINE`. (See page 153.)
  4. Compile all the functions into a DLL, using the commands appropriate for the platform and operating system.
    - ❖ Different platforms provide different mechanisms for exposing a function in a DLL; consult the system documentation for details.

See "The RVMS Data Type Source Code" on page 247 for an example.

## Registering a User-Defined Data Type to Attunity Connect

To register a user-defined data in the Attunity Connect environment, the `addon.def` file must include a section defining the data type:

1. Create a define file that includes either a data type “add-on” definition (`TYPE=DATATYPE`) or a startup function “add-on” definition (`TYPE=STARTUP`), specifying the name of the registration function as the `INIT-FUNCTION`. (For details, see "Registering a Startup Function" on page 151.)

The format for the “add-on” definition file is:

```
[name]
TYPE=DATATYPE or STARTUP
SHAREABLE-NAME=image
INIT-FUNCTION=function
```

where:

**name** – A unique identifier for the user-defined data type. This value must match the name that you specified for the data type in the `UDT_DATATYPE` structure (see "name" on page 291). If `TYPE` is `DATATYPE`, Attunity Connect locates the shareable image file (DLL) the first time a table containing a column of this data type is accessed. If `TYPE` is `STARTUP`, Attunity Connect locates the shareable image file (DLL) when Attunity Connect is started.

**image** – The path of the shareable image file (DLL on Windows platforms). If Attunity Connect cannot locate the DLL (for example, a specified environment variable or its translated filename does not exist), Attunity Connect writes an error message in the Attunity

Connect log file. See "System Notes" on page 253 for additional details about this entry.

- ❖ In this document, "DLL" is a generic term to refer to a shareable image or shared library of functions, as appropriate to the specific platform where Attunity Connect runs.

**function** – The name of the data type function in the DLL which passes a pointer to the UDT\_DATATYPE structure.

- ❖ The DLL must expose the INIT-FUNCTION in order for Attunity Connect to reference a data driver or data type. The symbol specified for the INIT-FUNCTION cannot be static. Different platforms provide different mechanisms for exposing a function in a DLL; consult the system documentation for details and see "System Notes" on page 253.

2. Use the ADDON option of the NAV\_UTIL utility to merge the define file with existing entries in addon.def. For details, see page 21.

## Data Type Functions

The Attunity Connect UDT API lets you define a new data type. To enable Attunity Connect to map the defined data type to one of the standard OLE DB/ODBC data types you must supply conversion routines as part of the data type definition. In this way, a user-defined data type is made available for displaying, sorting and any other operation that is permitted on the mapped data type.

This section describes the following functions:

- **Conversion Functions**
- **Comparison Functions**
- **Initialization/Assignment Functions**
- **Error Functions**

## Conversion Functions

This section describes the following functions:

- The TO\_STRING Function
- The FROM\_STRING Function
- The TO\_BASE Function
- The FROM\_BASE Function
- The CONVERT\_TO Function
- The TO\_EXPOSED Function
- The WIDTH\_TO\_SIZE Function

## The TO\_STRING Function

**Assignment** Specify the name of this function as the `to_string` member of the `UDT_DATATYPE` structure. (For details, see page 292.)

**Description** Attunity Connect calls this function to convert data stored in a user-defined data type to a string.

**Status** Required.

**Comments** The query processor requires conversion to standard (base) data types prior to evaluating any expression. If the data type definition does not include a direct conversion routine (`TO_BASE`), Attunity Connect uses a string as an intermediate step in converting a user-defined data type to its base data type.

For a description of possible return values, see "UDT\_STATUS" on page 306.

**Prototype** `UDT_STATUS (*UDT_FNC_TO_STRING) (`  
    `GDB_VAL_DESC *str_dsc,`  
    `GDB_VAL_DESC *udt_dsc);`

Parameter	Usage	Description
<code>str_dsc</code>	Output	A string data type, containing the converted value.
<code>udt_dsc</code>	Input	A user data type value, as returned by the driver in the <code>GDB_BUFFER</code> , that will be converted.  ❖ This function assumes an allocated buffer with a maximum size of 4K.

## The FROM\_STRING Function

**Assignment** Specify the name of this function as the `from_string` member of the `UDT_DATATYPE` structure. (For details, see page 293.)

**Description** Attunity Connect calls this function to convert data stored in a string to a user-defined data type.

**Status** Required.

**Comments** The query processor requires conversion to standard (base) data types prior to evaluating any expression. After evaluating the expression, the data must be converted back to its original type. If the data type definition does not include a direct conversion routine (`FROM_BASE`), Attunity Connect uses a string as an intermediate step in converting a user-defined data type from its base data type.

For a description of possible return values, see "UDT\_STATUS" on page 306.

**Prototype** UDT\_STATUS (\*UDT\_FNC\_FROM\_STRING) ( GDB\_VAL\_DESC \*udt\_dsc, GDB\_VAL\_DESC \*str\_dsc) ;

Parameter	Usage	Description
udt_dsc	Output	A user data type value, converted from a string.
str_dsc	Input	A string value to be converted. ❖ This function assumes a null terminated string that is "trimmed" of blanks.

## The TO\_BASE Function

**Assignment** Specify the name of this function as the **to\_base** member of the UDT\_DATATYPE structure. (For details, see page 293.)

**Description** Attunify Connect calls this function to map a user-defined data type value to its base type, as required by standard query operations.

**Status** Optional; however, both the TO\_BASE function and the FROM\_BASE function must be provided if either one is to be used.

**Comments** If direct mapping is possible between a user-defined data type and its base data type, such conversions generally perform better than conversions requiring an intermediate string step.

❖ You cannot define an atomic data type with a non-atomic base data type.

For a description of possible return values, see "UDT\_STATUS" on page 306.

**Prototype** UDT\_STATUS (\*UDT\_FNC\_TO\_BASE) ( GDB\_VAL\_DESC \*base\_dsc, GDB\_VAL\_DESC \*udt\_dsc) ;

Parameter	Usage	Description
base_dsc	Output	The converted value, stored in the data type indicated by the "base_id" for the user-defined data type.
udt_dsc	Input	A user-defined data type value, to be converted.

## The FROM\_BASE Function

**Assignment** Specify the name of this function as the **from\_base** member of the UDT\_DATATYPE structure. (For details, see page 293.)

**Description** Attunity Connect calls this function to restore a user-defined data type value from its base type, after performing standard query operations on a column that uses the user-defined data type.

**Status** Optional; however, both the TO\_BASE function and the FROM\_BASE function must be provided if either one is to be used.

**Comments** If direct mapping between a user-defined data type and its base data type is possible, such conversions generally perform better than conversions requiring an intermediate string step.

- ❖ You cannot define an atomic data type with a non-atomic base data type.

For a description of possible return values, see "UDT\_STATUS" on page 306.

**Prototype** UDT\_STATUS (\*UDT\_FNC\_FROM\_BASE) (  
    GDB\_VAL\_DESC \*udt\_dsc,  
    GDB\_VAL\_DESC \*base\_dsc) ;

Parameter	Usage	Description
udt_dsc	Output	The converted value, stored as the user-defined data type.
base_dsc	Input	The value to be converted, stored in the data type indicated by the "base_id" for the user-defined data type.

## The CONVERT\_TO Function

**Assignment** Specify the name of this function as the `convert_to` member of the UDT\_DATATYPE structure. (For details, see page 293.)

**Description** Attunity Connect calls this function to convert data of this data type to another data type.

**Status** Optional.

**Comments** Specify this function whenever direct mapping between a user-defined data type and another data type is possible. You can use this function in conjunction with the `convert_to_datatypes` member of the UDT\_DATATYPE structure, to specify mapping to several different data types. (For details, see page 294.)

If the function is not supplied or the target data type is not listed in the `convert_to_datatypes`, Attunity Connect attempts to use the TO\_BASE or TO\_STRING functions for the data type to do the conversion.

The function should return a value from the UDT\_STATUS enumeration, listed on page 306, indicating the status of the conversion.

**Prototype** `UDT_STATUS (*UDT_FNC_CONVERT_TO) (`  
`GDB_VAL_DESC *res_dsc,`  
`GDB_VAL_DESC *udt_dsc) ;`

Parameter	Usage	Description
<code>res_dsc</code>	Output	A user data type value for the result.
<code>udt_dsc</code>	Input	A pointer to the data to be converted.
Return Value	Usage	Description
<code>&lt;conversion status&gt;</code>	Output	The status of conversion: <code>UDT_OK</code> – First value ( <code>udt_dsc</code> ) was successfully converted to a value in the second data type ( <code>res_dsc</code> ). <code>UDT_NOT_OK</code> – The conversion could not be done. <code>UDT_OVERFLOW</code> – The conversion was done, but overflow was detected in the value of the resultant data type. <code>UDT_TRUNCATED</code> – The conversion was done, but truncation was detected in the value of the resultant data type.

## The TO\_EXPOSED Function

**Assignment** Specify the name of this function as the `to_exposed` member of the UDT\_DATATYPE structure. (For details, see page 294.)

**Description** This function accepts a descriptor of the “raw” data type in the physical file and a descriptor of the exposed data type. The function can then change the exposed data type size and width.

**Status** Optional.

**Comments** Specify this function if you have an application where the size occupied by the data in the physical file is different from the “exposed” size. Attunity Connect calls this function for every column of this data type in the table; however, only the width and scale of the exposed descriptor are used after the function is called.

Note the following:

- For an atomic data type, the size in the UDT\_DATATYPE structure refers to the physical storage size, not the exposed size.
- For a non-atomic data type, the SIZE clause specified in ADD refers to the physical storage, unless you supply a WIDTH\_TO\_SIZE function for the data type. (For details, see page 139.) The size of the exposed type is determined by what you set in the TO\_EXPOSED function.
- The string representation implemented in TO\_STRING and FROM\_STRING is the string representation of the exposed/base type, not the raw type. For example, when the physical data type is a binary string (such as "101") and the exposed type is byte (5 for "101"), the string representation is "5", not "101".

**Prototype** `unsigned long (*UDT_FNC_TO_EXPOSED) (`  
    `GDB_VAL_DESC *exposed_desc,`  
    `GDB_VAL_DESC *src_desc);`

---

Parameter	Usage	Description
-----------	-------	-------------

exposed_desc	Output	A descriptor of the exposed data type, with changes to the size and/or width of the data.
src_desc	Input	A descriptor of the “raw” data type in the physical file.

---

Return Value	Usage	Description
--------------	-------	-------------

<status>	Output	Zero if the function was unsuccessful in setting the data; non-zero if successful.
----------	--------	--

**Examples** ■ A user-defined data type is needed in order to access data stored as a 4-byte longword in the physical file, but exposed to the user as a 16 byte character string. The following is an example of such a function:

```
static long to_exposed(GDB_VAL_DESC *exposed_desc,
                      GDB_VAL_DESC *physical_desc)
{
    exposed_desc->width = 16;
    exposed_desc->scale = 0;
    return (1);
}
```

- A user-defined data type is needed in order to access data stored as a 2-byte word in the physical file, but exposed to the user as a 1 byte

character string. Change the above example to set the exposed\_desc->width to 1.

## The WIDTH\_TO\_SIZE Function

**Assignment** Specify the name of this function as the `width_to_size` member of the `UDT_DATATYPE` structure. (For details, see page 294.)

**Description** Attunity Connect calls this function for non-atomic data types, to calculate the number of bytes of physical storage occupied by the data type, based on the display width ("length" in `COLUMN_DA` structure).

**Status** Optional.

**Comments** You must set the `UDT_M_SIZED_` attribute for the data type when you populate the `UDT_DATATYPE` structure, as described on page 304.

When you describe a non-atomic field, either in ADD or in the `COLUMN_DA` metadata structure, you can specify only the display width (the `SIZE` clause in ADD or the `column_da->length` element). (For details, see page 276.) The number of bytes of physical storage is not necessarily equivalent to the display width, however, so this function uses the specified display width to determine the storage size of the physical buffer coming from the driver.

You can use this function in conjunction with the `max_display_length` member of the `UDT_DATATYPE` structure. (For details, see page 294.)

If the data type has a scale (`UDT_IS_SCALED` on page 304), Attunity Connect packs both the scale and width into one longword that serves as input to this function.

If this function is not supplied, Attunity Connect returns the width (`column_da->length`) of the data type as its physical size.

- ❖ If the size occupied by the data in the physical file is different from the "exposed" size, use the `TO_EXPOSED` function to specify the width and scale of the "exposed" data type. For details, see page 137.

**Prototype** `unsigned long (*UDT_FNC_WIDTH_TO_SIZE) (`  
`unsigned long width);`

Parameter	Usage	Description
<code>width</code>	Input	<p>The display width (<code>column_da-&gt;length</code>) of a user data type.</p> <ul style="list-style-type: none"> <li>❖ For a scaled data type, both the length and scale values are packed into this longword.</li> </ul>

Return Value	Usage	Description
<storage size>	Output	The number of bytes of physical storage occupied by the user data type.

- Examples** ■ A packed decimal can be specified in ADD with a clause such as:

```
DATATYPE IS DECIMAL SIZE IS 9 DIGITS
```

The physical size occupied by a packed decimal is determined by the formula:

$$(length/2) + 1$$

In this case width is 9 and size is 5.

- A packed decimal data type with a scale can be specified in ADD with a clause such as:

```
DATATYPE IS DECIMAL SIZE IS 9 DIGITS 3 FRACTIONS
```

Since Attunity Connect packs width and scale into one input parameter, you must decode the value as follows:

```
static long isam_decimal_width_to_size(
    INT4 width_scale)
{
    long width = width_scale >> 16;
    long scale = width_scale & 0x0000FFFF;
    return (calc_size(width, scale));
}
```

## Comparison Functions

This section describes the following functions:

- The COMPARE Function
- The IS\_EMPTY\_VAL Function
- The IS\_NULL\_VAL Function

### The COMPARE Function

**Assignment** Specify the name of this function as the **compare** member of the UDT\_DATATYPE structure. (For details, see page 293.)

**Description** The COMPARE function is called only if a column based on this data type is used in an index.

**Status** Optional.

**Comments** This function allows a “native” comparison between two columns of this data type, without intermediate mapping to strings. If not supplied, Attunity Connect uses the TO\_STRING function and compares data for this data type as strings.

- ❖ String comparisons of some data types may not always yield correct results.

By default, if the sort order for the raw data does not follow the sort order of the “exposed” data, comparisons are done using the exposed value. If you specify transformations between the physical data and the exposed type in the COMPARE function, you must use columns based on this type in an index to ensure that the correct rows are returned. Otherwise, you should supply a TO\_EXPOSED function to handle the necessary transformations for the data type regardless of whether its columns are used in an index.

**Prototype**

```
UDT_COMPARE_STATUS (*UDT_FNC_COMPARE) (
    GDB_VAL_DESC *udt1_dsc,
    GDB_VAL_DESC *udt2_dsc);
```

Parameter	Usage	Description
udt1_dsc	Input	A user data type value.
udt2_dsc	Input	A user data type value.
Return Value	Usage	Description
<compare status>	Output	The status of comparison:
		UDT_1ST_LESS_ – First value (udt1_dsc) is less than second value (udt2_dsc).
		UDT_EQUAL_ – The two supplied values are equal.
		UDT_1ST_GREATER_ – First value (udt1_dsc) is greater than second value (udt2_dsc).

## The IS\_EMPTY\_VAL Function

- Assignment** Specify the name of this function as the `is_empty_val` member of the UDT\_DATATYPE structure. (For details, see page 293.)
- Description** Attunity Connect calls this function to check whether a column in a buffer is the empty value of this data type.
- Status** Optional; however, both the IS\_EMPTY\_VAL function and the SET\_EMPTY\_VAL function must be provided if either one is to be used.
- Comments** If not supplied, Attunity Connect compares data for this data type against the default empty value of '\0'.

**Prototype** long (\*UDT\_FNC\_IS\_EMPTY\_VAL) (

```
    GDB_VAL_DESC *udt_dsc,
    void *val);
```

---

Parameter	Usage	Description
-----------	-------	-------------

udt_dsc	Input	A user data type value.
val	Input	A pointer to the data to be checked.

---

Return Value	Usage	Description
--------------	-------	-------------

<status>	Output	Zero if false; non-zero if true.
----------	--------	----------------------------------

## The IS\_NULL\_VAL Function

**Assignment** Specify the name of this function as the `is_null_val` member of the `UDT_DATATYPE` structure. (For details, see page 293.)

**Description** Attunity Connect calls this function to check whether a column in a buffer is set to the null value of this data type.

**Status** Optional; however, both the `IS_NULL_VAL` function and the `SET_NULL_VAL` function must be provided if either one is to be used.

**Comments** You must set the `UDT_M_NULLABLE_` attribute for the data type when you populate the `UDT_DATATYPE` structure, as described on page 304.

**Prototype** long (\*UDT\_FNC\_IS\_NULL\_VAL) (

```
    GDB_VAL_DESC *udt_dsc,
    void *val);
```

---

Parameter	Usage	Description
-----------	-------	-------------

udt_dsc	Input	A user data type value.
val	Input	A pointer to the data to be checked.

---

Return Value	Usage	Description
--------------	-------	-------------

<status>	Output	Zero if false; non-zero if true.
----------	--------	----------------------------------

## Initialization/Assignment Functions

This section describes the following functions:

- The `SET_EMPTY_VAL` Function
- The `SET_NULL_VAL` Function
- The `LOWEST_VAL` Function
- The `HIGHEST_VAL` Function

- The RANDOM\_VAL Function

## The SET\_EMPTY\_VAL Function

**Assignment** Specify the name of this function as the `set_empty_val` member of the `UDT_DATATYPE` structure. (For details, see page 293.)

**Description** Attunity Connect calls this function to set a column in a buffer to the empty value of this data type.

**Status** Optional; however, both the `IS_EMPTY_VAL` function and the `SET_EMPTY_VAL` function must be provided if either one is to be used.

**Comments** If a row is inserted and one of the columns is not specified, Attunity Connect calls this function to set the buffer with the “empty” value of this data type.

The “empty” value is not necessarily equivalent to the data type’s null value. (See “The SET\_NULL\_VAL Function” on page 143.) The empty value is retrieved “as is” in read operations.

If not supplied, Attunity Connect sets data for this data type to the default empty value of '\0'.

**Prototype**

```
long (*UDT_FNC_SET_EMPTY_VAL) (
    GDB_VAL_DESC *udt_dsc,
    void *val);
```

Parameter	Usage	Description
<code>udt_dsc</code>	Input	A user data type value.
<code>val</code>	Output	A pointer to the data to be set.

Return Value	Usage	Description
<code>&lt;status&gt;</code>	Output	Zero if the function was unsuccessful in setting the data; non-zero if successful.

## The SET\_NULL\_VAL Function

**Assignment** Specify the name of this function as the `set_null_val` member of the `UDT_DATATYPE` structure. (For details, see page 293.)

**Description** Attunity Connect calls this function to set a column in a buffer to the null value of this data type.

**Status** Optional; however, both the `IS_NULL_VAL` function and the `SET_NULL_VAL` function must be provided if either one is to be used.

**Comments** You must set the UDT\_M\_NULLABLE\_ attribute for the data type when you populate the UDT\_DATATYPE structure, as described on page 304.

**Prototype** `long (*UDT_FNC_SET_NULL_VAL) (`  
    `GDB_VAL_DESC *udt_dsc,`  
    `void *val);`

Parameter	Usage	Description
udt_dsc	Input	A user data type value.
val	Output	A pointer to the data to be set.

Return Value	Usage	Description
<status>	Output	Zero if the function was unsuccessful in setting the data; non-zero if successful.

## The LOWEST\_VAL Function

**Assignment** Specify the name of this function as the `lowest_val` member of the UDT\_DATATYPE structure. (For details, see page 294.)

**Description** Attunity Connect calls this function to set a column in a buffer to the lowest value for this data type.

**Status** Optional.

**Comments** If not supplied, Attunity Connect sets the data to point to 1001 bytes of 0 value.

**Prototype** `GDB_VAL_DESC * (*UDT_FNC_LOWEST_VAL) (`  
    `GDB_VAL_DESC *udt_dsc);`

Parameter	Usage	Description
udt_dsc	Output	A user data type to contain the lowest value.

Return Value	Usage	Description
<status>	Output	Zero if unsuccessful and the input parameter if successful.

## The HIGHEST\_VAL Function

**Assignment** Specify the name of this function as the `highest_val` member of the UDT\_DATATYPE structure. (For details, see page 293.)

**Description** Attunity Connect calls this function to set a column in a buffer to the highest value for this data type.

**Status** Optional.

**Comments** If not supplied, Attunity Connect sets the data to point to 1001 bytes of 0xff value.

**Prototype** GDB\_VAL\_DESC \* (\*UDT\_FNC\_HIGHEST\_VAL) (  
    GDB\_VAL\_DESC \*udt\_dsc);

Parameter	Usage	Description
udt_dsc	Output	A user data type to contain the highest value.

Return Value	Usage	Description
<status>	Output	Zero if unsuccessful and the input parameter if successful.

## The RANDOM\_VAL Function

**Assignment** Specify the name of this function as the `random_val` member of the UDT\_DATATYPE structure. (For details, see page 293.)

**Description** Attunity Connect calls this function to set a column in a buffer to a random value for this data type.

**Status** Optional.

**Comments** This function is useful when testing without real data.

If not supplied, Attunity Connect sets the data to point to 1001 bytes from a static area of memory. This default value does not change with subsequent calls.

**Prototype** GDB\_VAL\_DESC \* (\*UDT\_FNC\_RANDOM\_VAL) (  
    GDB\_VAL\_DESC \*udt\_dsc,  
    long num,  
    unsigned long add);

Parameter	Usage	Description
udt_dsc	Output	A user data type to contain the random value.
num	Input	A way to denote special cases such as: initialize/reset random number generator, set udt_dsc to highest value, and set udt_dsc to lowest value.
add	Input	A skew factor to use in producing a random number.

Return Value	Usage	Description
<status>	Output	Zero if unsuccessful and the input parameter if successful.

## Error Functions

This section describes the following function:

- The GET\_LAST\_ERROR Function

### The GET\_LAST\_ERROR Function

**Assignment** Specify the name of this function as the `get_last_error` member of the `UDT_DATATYPE` structure. (For details, see page 293.)

**Description** Attunity Connect calls this function whenever any of the data type functions returns an error code such as `UDT_NOT_OK_`. (See page 306.)

**Status** Optional.

**Comments** Attunity Connect writes the value of the string returned by this function to the log file. The string is also available to the client, so the data type routines can give the user textual information about errors. You should write the data type routines so that each column that uses a particular data type can be uniquely identified in the event of an error.

- ❖ The default log file (NAV.LOG, or NAVLOG on Tandem platforms) is located in the TMP directory under the directory where Attunity Connect is installed (on Tandem platforms, in the subvolume where Attunity Connect is installed). Use the `logFile` attribute in the Attunity Connect environment settings to specify a different file or location, as follows: `<debug logFile="pathname" >`.

Under OS/390, the string values are written to the job specified in the `StartupScript` parameter of the `WorkspaceXXX` section of the daemon settings.

**Prototype** `char * (*UDT_FNC_GET_LAST_ERROR) () ;`

Return Value	Usage	Description
<code>&lt;last error text&gt;</code>	Output	A pointer to a string with a description of to log file the last error to occur for this data type.

## Data Type Support Macros

The Developer SDK includes two macros in the UDT API that map to predefined support functions for data type processing:

- `UDT_CONVERT(A,B)`

Converts a value from one data type to a different type, using the support function `DT_CONVERT` (`nv_dt_convert`). For details, see page 154.

- `UDT_ADD_DATATYPE(A)`

Registers a user-defined data type to allow its use in Attunity Connect, using the support function `DT_ADD_DATATYPE` (`nv_dt_add_datatype`). For details, see page 154.



# **Part V**

## **Additional SDK Features**



# Chapter 9 Registering a Startup Function

The Developer SDK enables you to define a user function that you want to run when Attunity Connect initializes. This gives you an entry point from which you can execute any code for whatever purpose.

- ❖ One common use of startup functions is to register user-defined data types. (For details, see page 131.)

To register a startup function in the Attunity Connect environment, you must supply a definition in `addon.def`, as described in "Enabling the Developer SDK" on page 20.

- ❖ In an environment with multiple "add-on" drivers or data types, you should create a custom define file and merge it with `addon.def` using the `NAV_UTIL ADDON` option as described in "Defining Multiple "Add-Ons"" on page 21.

When Attunity Connect processes a startup function definition in the `addon.def` file, Attunity Connect immediately loads the DLL, searches the DLL for the named INIT-FUNCTION and calls the specified function.

The syntax for a startup function definition is:

```
[name]
TYPE=STARTUP
SHAREABLE-NAME=image
INIT-FUNCTION=function
```

where:

**name** – A unique identifier for the startup function. The *name* is referenced internally by Attunity Connect.

**image** – The path of the shareable image file (DLL). If Attunity Connect cannot locate the DLL (for example, a specified environment variable or its translated filename does not exist), Attunity Connect writes an error message in the Attunity Connect log file. See "System Notes" on page 253 for additional details about this entry.

- ❖ In this document, "DLL" is a generic term to refer to a shareable image or shared library of functions, as appropriate to the specific platform where Attunity Connect runs.

**function** – The name of a function in the DLL.

- ❖ The DLL must expose the INIT-FUNCTION in order for Attunity Connect to reference a data driver or data type. The symbol specified for the INIT-FUNCTION cannot be static. Different platforms provide different mechanisms for exposing a function in a DLL; consult the system documentation for details and see "System Notes" on page 253.

### Sample Startup Function

```
<<<----- Sample C Startup Function - Begin ----->>>

#ifndef WIN32
#define EXPORT_SYMBOL __declspec(dllexport)
#else
#define EXPORT_SYMBOL
#endif
static GDB_HELP_DEFINE;

EXPORT_SYMBOL mystartup_func(GDB_HELP_FUNCTIONS *help)
{
    gdb_help = help;

    /* User code goes here...
     * May use the GDB helper functions
     */
}

<<<----- Sample C Startup Function - End   --->>>
```

# Chapter 10 Developer SDK Predefined Functions

The Developer SDK includes a variety of support functions that you may find useful to perform common, low-level processing tasks in the custom code. For a user-defined data type definition, you must access these functions in order to register the data type in the Attunity Connect environment. For a data driver, you are not required to use these functions; however, Attunity Connect always passes the GDB\_HELP\_FUNCTIONS structure to the driver's LOAD function.

The Developer SDK defines and populates the GDB\_HELP\_FUNCTIONS structure to make these support functions available in the driver or user-defined data type code. (For details, see page 263.)

To use the support functions, perform the following tasks:

1. In the setup code for the custom library, use the following macro statement to establish a pointer to the predefined support functions structure (GDB\_HELP\_FUNCTIONS):

```
GDB_HELP_DEFINE;
```

If the code is split among more than one module, only the **first** module should use the above syntax to define a pointer to the support functions. Additional modules should point to the same definition, using the following macro statement:

```
GDB_HELP_USE;
```

2. In the INIT-FUNCTION, use the following macro statement to initialize the driver's GDB\_HELP\_FUNCTIONS pointer and set the version number of the appropriate function pointer:

```
GDB_INITIALIZE(fnc,help)
```

where:

**fnc** – pointer to structure containing function definitions, either UDT\_DATATYPE for a user defined data type, or GDB\_DB\_FUNCTIONS for a data driver.

**help** – GDB\_HELP\_FUNCTIONS argument to the INIT-FUNCTION of the DLL.

See the "Sample Data Drivers" on page 193 and "Sample RVMS Data Type" on page 247 for examples.

## Data Type Support Functions

The Developer SDK includes these support functions for use with user-defined data types:

- DT\_CONVERT
- DT\_ADD\_DATATYPE

### DT\_CONVERT

**Assignment** Attunity Connect specifies the name of this function as the **nv\_dt\_convert** member of the **GDB\_HELP\_FUNCTIONS** structure. (See page 263.)

**Description** This function converts a value from one data type to a different type.

Attunity Connect, like OLE DB, has a fixed set of data types and a set of well-defined interfaces. A user defined data type exists up to the level of the driver shell, but is converted to its exposed data type before the query processor evaluates any expressions based on that type. If the user-defined data type definition includes conversion routines for TO\_BASE and FROM\_BASE, this function calls those routines to perform the necessary conversions; otherwise, this function calls the TO\_STRING and FROM\_STRING routines.

**Prototype** `void (*GDB_FNC_DT_CONVERT) (`  
    `GDB_VAL_DESC *target_desc,`  
    `GDB_VAL_DESC *source_desc);`

Parameter	Usage	Description
<code>target_desc</code>	Input/O utput	Pointer to a structure that identifies the desired data type for the output; the converted data is stored in this structure.
<code>source_desc</code>	Input	Pointer to structure that contains data to be converted, stored using a specified data type.

**Relevant Macro** `UDT_CONVERT (A, B)`

For details, see page 147.

### DT\_ADD\_DATATYPE

**Assignment** Attunity Connect specifies the name of this function as the **nv\_dt\_add\_datatype** member of the **GDB\_HELP\_FUNCTIONS** structure. (See page 263.)

**Description** This function registers a user-defined data type to allow its use in Attunity Connect.

---

**Prototype** long (\*GDB\_FNC\_DT\_ADD\_DATATYPE)  
(UDT\_DATATYPE \*datatype);

Parameter	Usage	Description
datatype	Input	A UDT_DATATYPE structure that you have populated.

**Relevant Macro** UDT\_ADD\_DATATYPE (A)

For details, see page 147.

## Memory Pool Functions

These support functions make it easier to manage memory usage in the Attunity Connect environment:

- MEMP\_NEW
- MEMP\_MALLOC
- MEMP\_MALLOC8
- MEMP\_FREE
- MEMP\_FREE\_POOL
- MEMP\_DELETE

### MEMP\_NEW

**Assignment** Attunity Connect specifies the name of this function as the `nv_memp_new` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function creates a new memory pool.

**Prototype** void \* (\*GDB\_FNC\_MEMP\_NEW) (char \*memp\_name,  
long initial\_size);

Parameter	Usage	Description
memp_name	Input	Name of memory pool to be created <i>For Future Use:</i> This could be printed as part of a memory usage report.
initial_size	Input	Size (in bytes) of memory pool to be created.

## MEMP\_MALLOC

**Assignment** Attunity Connect specifies the name of this function as the `nv_memp_malloc` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function allocates memory from a previously created pool.

**Prototype** `void * (*GDB_FNC_MEMP_MALLOC) (long size,  
void *memp);`

Parameter	Usage	Description
size	Input	Number of bytes to be allocated.
memp	Input	Pointer to memory pool.

## MEMP\_MALLOC8

**Assignment** Attunity Connect specifies the name of this function as the `nv_memp_malloc8` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function allocates memory aligned on an 8-byte boundary, from an existing pool, as required by certain platforms and operating systems.

**Prototype** `void * (*GDB_FNC_MEMP_MALLOC8) (long size,  
void *memp);`

Parameter	Usage	Description
size	Input	Number of bytes to be allocated.
memp	Input	Pointer to memory pool.

## MEMP\_FREE

❖ *For Future Use*

This function will be implemented in an upcoming version of the Developer SDK.

**Assignment** Attunity Connect specifies the name of this function as the `nv_memp_free` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function frees a portion of allocated memory by returning a pointer to an available location in the pool.

---

**Prototype** long (\*GDB\_FNC\_MEMP\_FREE) (void \*memp);

Parameter	Usage	Description
memp	Input	Pointer to memory pool.

## MEMP\_FREE\_POOL

**Assignment** Attunity Connect specifies the name of this function as the `nv_memp_free_pool` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function flushes the memory pool, freeing any memory allocated from the pool but leaving the pool available for further use.

**Prototype** long (\*GDB\_FNC\_MEMP\_FREE\_POOL) (void \*memp);

Parameter	Usage	Description
memp	Input	Pointer to memory pool to be flushed.

## MEMP\_DELETE

**Assignment** Attunity Connect specifies the name of this function as the `nv_memp_delete` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function frees any memory allocated from the pool and deletes the memory pool.

**Prototype** long (\*GDB\_FNC\_MEMP\_DELETE) (void \*memp);

Parameter	Usage	Description
memp	Input	Pointer to memory pool to be deleted.

## Dynamic Array Functions

These support functions enable you to manipulate dynamic arrays in the Attunity Connect environment:

- ARR\_NEW
- ARR\_RESET
- ARR\_SIZE
- ARR\_DELETE
- ARR\_INSERT
- ARR\_ADD
- ARR\_SET\_ENTRY
- ARR\_COPY

- ARR\_FREE
- COMPARE
- ARR\_BINSEARCH
- ARR\_BSEARCH

## ARR\_NEW

**Assignment** Attunity Connect specifies the name of this function as the `nv_arr_new` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function allocates a new array, using an existing memory pool.

```
Prototype long (*GDB_FNC_ARR_NEW) (
    long size,
    GDB_DYNARRAY *array,
    long extent,
    void *memp);
```

Parameter	Usage	Description
size	Input	Number of elements to be created in array
array	Input	Address of pointer to array to be created. ❖ GDB_DYNARRAY is the address of an array of pointers. The address may be changed by the function, for example, if the array needs to be reallocated to accommodate additional entries.
extent	Input	
memp	Input	Pointer to memory pool from which array is allocated.

## ARR\_RESET

**Assignment** Attunity Connect specifies the name of this function as the `nv_arr_reset` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function marks all entries in an array as unused, but does not free memory.

```
Prototype long (*GDB_FNC_ARR_RESET) (GDB_DYNARRAY *array);
```

Parameter	Usage	Description
array	Input	Address of pointer to array to be reset. ❖ GDB_DYNARRAY is the address of an array of pointers. The address may be changed, for example, if the array needs to be reallocated to accommodate additional entries.

## ARR\_SIZE

**Assignment** Attunity Connect specifies the name of this function as the `nv_arr_size` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function returns a count of the elements actually in the array. The number of entries allocated for the array can be larger than this size.

**Prototype** `long (*GDB_FNC_ARR_SIZE) (GDB_DYNARRAY *array);`

Parameter	Usage	Description
array	Input	Pointer to array to be created. <ul style="list-style-type: none"> <li>❖ <code>GDB_DYNARRAY</code> is the address of an array of pointers. The address may be changed by the function, for example, if the array needs to be reallocated to accommodate additional entries.</li> </ul>
<b>Return Value Description</b>		
<array size>		Count of elements actually in the array.

## ARR\_DELETE

**Assignment** Attunity Connect specifies the name of this function as the `nv_arr_delete` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function deletes a specified entry from the array.

**Prototype** `long (*GDB_FNC_ARR_DELETE) (GDB_DYNARRAY *array,  
long entry);`

Parameter	Usage	Description
array	Input	Address of pointer to array. <ul style="list-style-type: none"> <li>❖ <code>GDB_DYNARRAY</code> is the address of an array of pointers. The address may be changed, for example, if the array needs to be reallocated to accommodate additional entries.</li> </ul>
entry	Input	Index number of element to be deleted from array.

## ARR\_INSERT

**Assignment** Attunity Connect specifies the name of this function as the `nv_arr_insert` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function inserts an entry at a specific location in the array. Before inserting the entry, you must determine the correct entry location for

the insertion. For a sorted array, Attunity Connect provides the support function ARR\_BINSEARCH which you can call to determine the location to be used in order to maintain properly sorted information in the array.

**Prototype** long (\*GDB\_FNC\_ARR\_INSERT) (GDB\_DYNARRAY \*array,  
                  long entry);

Parameter	Usage	Description
array	Input	Address of pointer to array.  ❖ GDB_DYNARRAY is the address of an array of pointers. The address may be changed, for example, if the array needs to be reallocated to accommodate additional entries.
entry	Input	Index number of entry location for insertion.

## ARR\_ADD

**Assignment** Attunity Connect specifies the name of this function as the **nv\_arr\_add** member of the **GDB\_HELP\_FUNCTIONS** structure. (See page 263.)

**Description** This function allocates a new entry at the end of the array, returning an index into the array. If the existing array is not large enough to accommodate the new entry, the function:

1. Creates a new array.
2. Allocates needed space.
3. Copies data from the existing array into the new array.
4. Destroys the old (too small) array.

**Prototype** long (\*GDB\_FNC\_ARR\_ADD) (GDB\_DYNARRAY \*array);

Parameter	Usage	Description
array	Input	Address of pointer to array.  ❖ GDB_DYNARRAY is the address of an array of pointers. The address may be changed, for example, if the array needs to be reallocated to accommodate additional entries.

Return Value	Description
<index entry>	Index number into array.

## ARR\_SET\_ENTRY

**Assignment** Attunity Connect specifies the name of this function as the `nv_arr_set_entry` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function sets a value at a specific entry location in the array. If necessary, the function allocates null entries up to the specified number in the array.

**Prototype**

```
long (*GDB_FNC_ARR_SET_ENTRY) (
    GDB_DYNARRAY *array,
    long entry,
    void *value);
```

Parameter	Usage	Description
array	Input	Address of pointer to array. ❖ GDB_DYNARRAY is the address of an array of pointers. The address may be changed, for example, if the array needs to be reallocated to accommodate additional entries.
entry	Input	Index number where entry is written in array.
value	Input	Pointer to value that is written for entry.

## ARR\_COPY

**Assignment** Attunity Connect specifies the name of this function as the `nv_arr_copy` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function copies the entries from one array to another.

- ❖ This function is used internally by ARR\_ADD when it needs to extend an array beyond its specified size.

**Prototype**

```
long (*GDB_FNC_ARR_COPY) (
    GDB_DYNARRAY *array_to,
    GDB_DYNARRAY *array_from,
    void *memp);
```

Parameter	Usage	Description
array_to	Output	Address of pointer to new array (with copied data).

Parameter	Usage	Description
array_from	Input	Address of pointer to existing array that is copied. <ul style="list-style-type: none"><li>❖ GDB_DYNARRAY is the address of an array of pointers. The address may be changed, for example, if the array needs to be reallocated to accommodate additional entries.</li></ul>
memp	Input	Pointer to memory pool from which new array is allocated.

## ARR\_FREE

**Assignment** Attunity Connect specifies the name of this function as the **nv\_arr\_free** member of the **GDB\_HELP\_FUNCTIONS** structure. (See page 263.)

**Description** This function frees any memory allocated for the array **and** deletes the array.

**Prototype** `long (*GDB_FNC_ARR_FREE)(GDB_DYNARRAY *array);`

Parameter	Usage	Description
array	Input	Address of pointer to array to be destroyed. <ul style="list-style-type: none"><li>❖ GDB_DYNARRAY is the address of an array of pointers. The address may be changed, for example, if the array needs to be reallocated to accommodate additional entries.</li></ul>

## COMPARE

**Assignment** Attunity Connect specifies the name of this function as the **nv\_compare** member of the **GDB\_HELP\_FUNCTIONS** structure. (See page 263.)

**Description** This function checks the supplied key value against the data in the array, and returns a value indicating whether the key is greater than, less than, or equal to the entry data.

- ❖ This function is used internally by ARR\_BINSEARCH and ARR\_BSEARCH to manage sorted arrays.

**Prototype** `long (*GDB_FNC_COMPARE)(  
    void *key, void *entry);`

Parameter	Usage	Description
key	Input	Key value for comparison.
entry	Input	Index number of entry whose data is to be compared with key value.

Return Value	Description
<compare status>	-1 if key value is greater than data in entry 0 if key value is equal to data in entry 1 if key value is less than data in entry.

## ARR\_BINSEARCH

**Assignment** Attunity Connect specifies the name of this function as the **nv\_arr\_binsearch** member of the **GDB\_HELP\_FUNCTIONS** structure. (See page 263.)

**Description** This function operates on a sorted array to determine the location that should be used in order to maintain properly sorted information when inserting a new value into the array. Based on the results of the **COMPARE** function, you can decide whether to insert the new entry using **ARR\_INSERT** or update the existing entry whose value matches the supplied key.

- ❖ This function can be used for any sorted array, not just dynamic arrays.

**Prototype**

```
long (*GDB_FNC_ARR_BINSEARCH) (
    void **array,
    void *key,
    long array_size,
    long *entry,
    GDB_FNC_COMPARE compare_fnc);
```

Parameter	Usage	Description
array	Input	Pointer to array ( <b>not</b> address of pointer).
key	Input	Key value for comparison.
array_size	Input	Count of elements actually in the array.
entry	Output	If key value is not found in array, entry is the index number of location where value should be inserted.  If key value is found, entry is the index number where the value was found.
compare_fnc	Input	Name of function to be used when comparing key value to array entries, to locate insertion point.

Return Value	Description
<insert status>	0 if key value is not found in array. Any other value indicates key value was found.

## ARR\_BSEARCH

**Assignment** Attunity Connect specifies the name of this function as the **nv\_arr\_bsearch** member of the **GDB\_HELP\_FUNCTIONS** structure. (See page 263.)

**Description** This function performs a binary search on a sorted array, comparing the array entries to a supplied search key value.

- ❖ This function is used internally by ARR\_BINSET to determine the entry location of a value to be inserted into a sorted array. This function can be used for any sorted array, not just dynamic arrays.

**Prototype**

```
long (*GDB_FNC_ARR_BSEARCH) (
    void **array,
    void *search_key,
    long array_size,
    GDB_FNC_COMPARE compare_fnc);
```

Parameter	Usage	Description
array	Input	Pointer to array ( <b>not</b> address of pointer).
search_key	Input	Key value for comparison.
array_size	Input	Count of elements actually in the array.
compare_fnc	Input	Name of function to be used when searching for key value in array entries.

Return Value	Description
<search status>	Index number where key value was found, or -1 if key value is not found in array.

## Dynamic String Functions

These support functions enable you to manipulate dynamic strings in the Attunity Connect environment:

- DSTR\_NEW
- DSTR\_APPEND
- DSTR\_M\_APPEND
- DSTR\_RESET
- DSTR\_LENGTH
- DSTR\_INSERT
- DSTR\_CUT
- DSTR\_GET\_ITEM

## DSTR\_NEW

**Assignment** Attunity Connect specifies the name of this function as the `nv_dstr_new` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function allocates a new dynamic string, using a previously created memory pool.

**Prototype**

```
void (*GDB_FNC_DSTR_NEW) (
    char **dstr,
    char *initial_str,
    long initial_size,
    long increment_size,
    void *memp);
```

Parameter	Usage	Description
<code>dstr</code>	Input	Address of dynamic string to be created. (Passing by address allows for growth of string.)
<code>initial_str</code>	Input	Initial data to be stored in dynamic string.
<code>initial_size</code>	Input	Number of bytes in initial data string.
<code>increment_size</code>	Input	Number of bytes for each subsequent increment in dynamic string.
<code>memp</code>	Input	Pointer to memory pool from which dynamic string is allocated.

## DSTR\_APPEND

**Assignment** Attunity Connect specifies the name of this function as the `nv_dstr_append` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function appends data to a previously created dynamic string.

**Prototype**

```
void (*GDB_FNC_DSTR_APPEND) (
    char **dstr,
    char *append_str,
    long append_str_size);
```

Parameter	Usage	Description
<code>dstr</code>	Input	Address of dynamic string to which data is appended.
<code>append_str</code>	Input	Data to be appended into dynamic string.
<code>append_str_size</code>	Input	Number of bytes in dynamic string. If set to -1, function assumes string is null terminated and determines size by itself.

## DSTR\_M\_APPEND

**Assignment** Attunity Connect specifies the name of this function as the **nv\_dstr\_m\_append** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Description** This function appends multiple strings to a previously created dynamic string.

**Prototype** `void (*GDB_FNC_DSTR_M_APPEND)(char **dstr, ...);`

Parameter	Usage	Description
dstr	Input	Address of dynamic string to which data is appended. The function supports a varying number of parameters, but each string must be null terminated, followed by NULL to terminate the list.

## DSTR\_RESET

**Assignment** Attunity Connect specifies the name of this function as the **nv\_dstr\_reset** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Description** This function resets a dynamic string by inserting a NULL at the beginning of the string, but does not free memory.

**Prototype** `void (*GDB_FNC_DSTR_RESET)(char *dstr);`

Parameter	Usage	Description
dstr	Input	Pointer to dynamic string that is reset.

## DSTR\_LENGTH

**Assignment** Attunity Connect specifies the name of this function as the **nv\_dstr\_length** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Description** This function returns the current length of dynamic string.

**Prototype** `long (*GDB_FNC_DSTR_LENGTH)(char *dstr);`

Parameter	Usage	Description
dstr	Input	Pointer to dynamic string.

Return Value	Description
<length>	Actual length of data in dynamic string. (Allocated size can be longer than the current length.)

## DSTR\_INSERT

- Assignment** Attunity Connect specifies the name of this function as the `nv_dstr_insert` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)
- Description** This function inserts a string into a specified location within a dynamic string.

```
Prototype void (*GDB_FNC_DSTR_INSERT) (
    char **dstr,
    long place,
    char *insert_str,
    long insert_str_size);
```

Parameter	Usage	Description
dstr	Input	Address of dynamic string in which data is inserted.
place	Input	Location in dynamic string where insertion should begin.
insert_str	Input	Data to be inserted in dynamic string.
insert_str_size	Input	Number of bytes to be inserted.

## DSTR\_CUT

- Assignment** Attunity Connect specifies the name of this function as the `nv_dstr_cut` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)
- Description** This function cuts a number of bytes from a specified location within a dynamic string.

```
Prototype void (*GDB_FNC_DSTR_CUT) (
    char *dstr,
    long place,
    long cut_size);
```

Parameter	Usage	Description
dstr	Input	Address of dynamic string to be cut.
place	Input	Location in dynamic string where deletion begins.
cut_size	Input	Number of bytes to be cut from dynamic string.

## DSTR\_GET\_ITEM

**Assignment** Attunity Connect specifies the name of this function as the `nv_dstr_get_item` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function searches a delimited string for a selected item.

You can use this function to parse elements of `DB_COMMAND` metadata for various structures within a data driver.

**Prototype** `char * (*GDB_FNC_DSTR_GET_ITEM) (`  
    `char *str,`  
    `long item_num,`  
    `char delimiter,`  
    `void *memp);`

Parameter	Usage	Description
<code>str</code>	Input	String to be searched.
<code>item_num</code>	Input	Number of the item to be located, starting with zero.
<code>delimiter</code>	Input	String that delimits items in list.
<code>memp</code>	Input	Pointer to memory pool to use for copy of string item.

## General String Functions

The Developer SDK includes the following functions for general string operations:

- `STR_C_STRING`
- `STR_C_STRING_PAD`
- `STR_STRCONCAT`

## STR\_C\_STRING

**Assignment** Attunity Connect specifies the name of this function as the `nv_str_c_string` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function copies a string into the memory pool.

**Prototype** `char * (*GDB_FNC_STR_C_STRING)`  
    `(char *str, long size, void *memp);`

Parameter	Usage	Description
<code>str</code>	Input	Data string to be copied into memory pool.

Parameter	Usage	Description
size	Input	Number of bytes to be copied. If size=-1, function assumes string is null terminated.
memp	Input	Pointer to memory pool to which string is copied.

## STR\_C\_STRING\_PAD

**Assignment** Attunity Connect specifies the name of this function as the `nv_str_c_string_pad` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function copies a string into the memory pool, padding with blanks up to the specified size.

**Prototype** `char * (*GDB_FNC_STR_C_STRING_PAD)(char *str, long size, void *memp);`

Parameter	Usage	Description
str	Input	Data string to be copied into memory pool.
size	Input	Number of bytes to be copied. If size is greater than the length of the string, function pads copied value with trailing blanks, up to the specified size.
memp	Input	Pointer to memory pool to which string is copied.

## STR\_STRCONCAT

**Assignment** Attunity Connect specifies the name of this function as the `nv_str_strconcat` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function concatenates any number of data strings.

**Prototype** `char * (*GDB_FNC_STR_STRCONCAT)(char *str,...);`

Parameter	Usage	Description
str	Input	Data strings to be concatenated. You can supply multiple strings, followed by a NULL to terminate the list.  The buffer is pre-allocated, so the user is responsible for allocating a large enough string to accommodate the concatenation operations.

## Filter Functions

### TRAVERSE\_FILTER

**Assignment** Attunity Connect specifies the name of this function as the `nv_traverse_filter` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function is supplied to help traverse the tree.

The user function is called in the right order, for every operand and its parameters return a handler. Operators like AND or OR are called with the handlers returned by previous calls.

For details about the arguments in this function, see the descriptions of the `GDB_FTREE_NODE` structure on page 284 and the `SET_FILTER` function on page 48.

For a description of possible return values, see "GDB\_STATUS" on page 305.

```
Prototype GDB_STATUS (*GDB_FNC_TRAVERSE_FILTER) (
    GDB_FTREE_HANDLE tree_handle,
    GDB_FTREE_NODE_HANDLE *root_node_handle,
    long n_filter_tree,
    long *filter_tree,
    long n_filter_consts,
    GDB_VAL_DESC **filter_consts,
    GDB_FNC_PROCESS_FTREE_NODE process_ftree_node);
```

## Metadata Support Functions

For a data source that uses fixed metadata, it may be more efficient to process metadata definitions from a C header file than from repeated checks to the ADD. You can use the following support functions to provide a link between a header file and the driver metadata routines:

- `GDBH_DESCRIBE_TABLE`
- `GDBH_DESCRIBE_SP`
- `GDBH_GET_ITEM_LIST`
- `GDB_GET_TABLE_DA`

To produce a header file from ADD metadata, run `NAV_UTIL` with the `ADDL_TO_GDBH` option.

## GDBH\_DESCRIBE\_TABLE

**Assignment** Attunity Connect specifies the name of this function as the `nv_gdbh_describe_table` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function can be called from the driver's `DESCRIBE_TABLE` function, to access a table's metadata definition in a header file.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDBH_FNC_DESCRIBE_TABLE) (
    char *table_name,
    TABLE_DA **table_da,
    GDBH_ROOT_TABLE *gdbh_root);
```

Parameter	Usage	Description
table_name	Input	Name of the table whose metadata you want to retrieve.
table_da	Output	Address of a "TABLE_DA" structure populated with the metadata for the specified table.
gdbh_root	Input	Pointer to table's metadata, as defined in the header file.

## GDBH\_DESCRIBE\_SP

**Assignment** Attunity Connect specifies the name of this function as the `nv_gdbh_describe_sp` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function can be called from the driver's `DESCRIBE_SP` function, to access a stored procedure's metadata definition in a header file.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDBH_FNC_DESCRIBE_SP) (
    char *table_name,
    TABLE_DA **table_da,
    GDBH_ROOT_TABLE *gdbh_root);
```

Parameter	Usage	Description
table_name	Input	Name of the stored procedure whose metadata you want to retrieve.

Parameter	Usage	Description
table_da	Output	Address of a "TABLE_DA" structure populated with the metadata for the specified stored procedure.
gdbh_root	Input	Pointer to stored procedure's metadata, as defined in the header file.

## GDBH\_GET\_ITEM\_LIST

- Assignment** Attunity Connect specifies the name of this function as the `nv_gdbh_get_item_list` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)
- Description** This function can be called from the driver's `GET_ITEM_LIST` function, to obtain a list of tables and procedures whose metadata is defined in a header file. (See page 39.)

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype**

```
GDB_STATUS (*GDBH_FNC_GET_ITEM_LIST) (
    GDB_ITEM_DESC **item_list,
    GDB_LIST_REQUEST request,
    NAD_GDBH_ITEM_DESC *gdbh_item_desc,
    GDB_ITEM_TYPE item_type,
    char *mask,
    GDBH_ROOT_TABLE *gdbh_root);
```

Parameter	Usage	Description
item_list	Output	A pointer to a "GDB_ITEM_DESC" structure with the name and description of a table or procedure.
request	Input	The type of request needed in the list operation, either <code>GDB_ITEM_LIST_OPEN_</code> , <code>GDB_ITEM_LIST_GET_</code> or <code>GDB_ITEM_LIST_CLOSE_</code> .
gdbh_item_desc	Input	Pointer to a structure that describes the item.
item_type	Input	The type of item to be listed. For this support function, the only valid types are <code>GDB_ITEM_TABLE_</code> (table) and <code>GDB_ITEM_SP_</code> (stored procedure).

Parameter	Usage	Description
mask	Input	A mask with wildcards on the required table names. If the driver ignores this mask, Attunify Connect uses the mask to filter out the tables that do not match it. Attunify Connect supports the following wildcards: <b>asterisk (*)</b> – multiple characters. <b>question mark (?)</b> – a single character.
gdbh_root	Input	Pointer to fixed metadata, as defined in the header file.

## GDB\_GET\_TABLE\_DA

**Assignment** Attunify Connect specifies the name of this function as the `nv_gdb_get_table_da` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function can be called from the driver's `GET_ITEM_LIST` function following a call to `GDBH_GET_ITEM_LIST`, to access metadata from the data source (TDP) for a specified table/procedure.

For a description of possible return values, see "GDB\_STATUS" on page 305.

**Prototype** `GDB_STATUS (*GDB_FNC_GET_TABLE_DA) (`  
`TABLE_DA ** table_da,`  
`char *item_name,`  
`GDB_ITEM_TYPE item_type,`  
`char *tdp_name);`

Parameter	Usage	Description
table_da	Output	Address of a "TABLE_DA" structure populated with the metadata for the specified table.
item_name	Input	Name of the item whose metadata you want to retrieve.
item_type	Input	The type of item to be retrieved. For this support function, the only valid types are <code>GDB_ITEM_TABLE_</code> (table) and <code>GDB_ITEM_SP_</code> (stored procedure).
tdp_name	Input	Name of data source from which the item is retrieved.

## NLS Functions

❖ *For Future Use:*

These functions will be implemented in an upcoming version of the Developer SDK.

- NLS\_CVTFNC
- RESET
- MBLEN
- CP\_INFO
- CP\_STRCHR
- CP\_STRSTR
- CP\_STRICMP
- CP\_UPPERCASE
- CP\_STRNWCMP
- CP\_STRNICMP
- CP\_TOUPPER
- CP\_LOWER CASE

### NLS\_CVTFNC

**Assignment** Attunity Connect specifies the name of this function as the `nv_nls_cvtfnc` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype** `long (*NLS_CVTFNC) (`  
    `unsigned char *,`  
    `long, unsigned char *,`  
    `long, long);`

### RESET

**Assignment** Attunity Connect specifies the name of this function as the `nv_reset` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype** `void (*RESET) (struct NLS_CPDEF *);`

### MBLEN

**Assignment** Attunity Connect specifies the name of this function as the `nv_mblen` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype** `long (*MBLEN) (unsigned char*, NLS_CTX*);`

## CP\_INFO

**Assignment** Attunity Connect specifies the name of this function as the **nv\_cp\_info** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Prototype** `unsigned char * (*CP_INFO) (long);`

## CP\_STRCHR

**Assignment** Attunity Connect specifies the name of this function as the **nv\_cp\_strchr** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Prototype** `unsigned char * (*CP_STRCHR) (`  
    `unsigned char *target,`  
    `unsigned char c);`

## CP\_STRSTR

**Assignment** Attunity Connect specifies the name of this function as the **nv\_cp strstr** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Prototype** `unsigned char * (*CP_STRSTR) (`  
    `unsigned char *str1,`  
    `unsigned char *str2);`

## CP\_STRICTCMP

**Assignment** Attunity Connect specifies the name of this function as the **nv\_cp\_strcmp** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Prototype** `long (*CP_STRICTCMP) (`  
    `unsigned char *str1,`  
    `unsigned char *str2);`

## CP\_UPPERCASE

**Assignment** Attunity Connect specifies the name of this function as the **nv\_cp\_uppercase** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Prototype** `unsigned char * (*CP_UPPERCASE) (`  
    `unsigned char *obj,`  
    `unsigned char *src);`

## CP\_STRNWCMP

**Assignment** Attunity Connect specifies the name of this function as the `nv_cp_strnwcmp` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype** `long (*CP_STRNWCMP) (`  
    `unsigned char *string,`  
    `unsigned char *mask, long max_len);`

## CP\_STRNICMP

**Assignment** Attunity Connect specifies the name of this function as the `nv_cp_strnicmp` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype** `long (*CP_STRNICMP) (`  
    `unsigned char *str1,`  
    `unsigned char *str2, long len);`

## CP\_TOUPPER

**Assignment** Attunity Connect specifies the name of this function as the `nv_cp_toupper` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype** `unsigned char (*CP_TOUPPER) (unsigned char c);`

## CP\_LOWERCASE

**Assignment** Attunity Connect specifies the name of this function as the `nv_cp_lowercase` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype** `unsigned char *(*CP_LOWERCASE) (`  
    `char *obj,`  
    `char *src);`

## Code Page Support Functions

- ❖ *For Future Use:*  
These functions will be implemented in an upcoming version of the Developer SDK.
- CP\_REGISTER\_CVT
- CP\_GET\_ID
- CP\_REGISTER

## CP\_REGISTER\_CVT

**Assignment** Attunity Connect specifies the name of this function as the `nv_cp_register_cvt` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype**

```
long (*GDB_FNC_CP_REGISTER_CVT) (
    long cp1_id,
    long cp2_id,
    NLS_CVTFNC cp1_cp2_cvt_fnc,
    NLS_CVTFNC cp2_cp1_cvt_fnc);
```

## CP\_GET\_ID

**Assignment** Attunity Connect specifies the name of this function as the `nv_cp_get_id` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype**

```
long (*GDB_FNC_CP_GET_ID) (
    char *codepage_name);
```

## CP\_REGISTER

**Assignment** Attunity Connect specifies the name of this function as the `nv_cp_register` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Prototype**

```
long (*GDB_FNC_CP_REGISTER) (
    char *codepage_name,
    void *codepage_routine_v);
```

## Profile Functions

A “profile” file is written in a generic format, and includes one or more entries that define elements of the environment. For example, the Developer SDK definition file (`addon.def`).

In the Attunity Connect profile files, the format of an entry is a section name enclosed by square brackets (under OS/390 the section names are enclosed in angled brackets), followed by one or more lines with items assigned values, for example:

```
[SECTION]
ITEM=value
ITEM2=value2
```

Empty lines (or lines containing only blank characters) are ignored when the profile file is parsed or processed. Section and item names are case-insensitive, but values are case-sensitive.

The following read-only support functions enable you to read and parse a file that is written in the Attunity Connect generic profile format:

- GDB\_PROF\_INIT\_FILE
- GDB\_PROF\_SECTION\_FIRST
- GDB\_PROF\_SECTION\_NEXT
- GDB\_PROF\_ITEM\_FIRST
- GDB\_PROF\_ITEM\_NEXT
- GDB\_PROF\_GET\_VALUE
- GDB\_PROF\_GET\_CUR\_VALUE
- GDB\_PROF\_SET\_SECTION
- GDB\_PROF\_SET\_ITEM
- GDB\_PROF\_DELETE

## GDB\_PROF\_INIT\_FILE

**Assignment** Attunity Connect specifies the name of this function as the `nv_prof_init_file` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function opens the file to be parsed.

**Prototype** `void* (*GDB_FNC_PROF_INIT_FILE)(char *file);`

Parameter	Usage	Description
file	Input	Name of the file to be parsed.

Return Value	Description
<prof>	Pointer to file, to be used in the other profile functions. On error, the function returns NULL.

## GDB\_PROF\_SECTION\_FIRST

**Assignment** Attunity Connect specifies the name of this function as the `nv_prof_section_first` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function sets the current section to the first section in the file.

**Prototype** `char* (*GDB_FNC_PROF_SECTION_FIRST)(  
void *prof);`

Parameter	Usage	Description
<code>prof</code>	Input	Pointer to the file being parsed.
Return Value	Description	
<code>&lt;section&gt;</code>	Section name, or NULL if there are no sections in the file.	

## GDB\_PROF\_SECTION\_NEXT

**Assignment** Attunity Connect specifies the name of this function as the `nv_prof_section_next` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function sets the current section to the next section in the file.

**Prototype** `char* (*GDB_FNC_PROF_SECTION_NEXT)(  
void *prof);`

Parameter	Usage	Description
<code>prof</code>	Input	Pointer to the file being parsed.
Return Value	Description	
<code>&lt;section&gt;</code>	Section name, or NULL if there are no more sections in the file.	

## GDB\_PROF\_ITEM\_FIRST

**Assignment** Attunity Connect specifies the name of this function as the `nv_prof_item_first` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function sets the current item to the first item in the current section.

**Prototype** `char* (*GDB_FNC_PROF_ITEM_FIRST)(  
void *prof);`

Parameter	Usage	Description
<code>prof</code>	Input	Pointer to the file being parsed.
Return Value	Description	
<code>&lt;item&gt;</code>	Item name, or NULL if there are no items in the section.	

## GDB\_PROF\_ITEM\_NEXT

**Assignment** Attunity Connect specifies the name of this function as the `nv_prof_item_next` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function sets the current item to the next item in the current section.

**Prototype** `char* (*GDB_FNC_PROF_ITEM_NEXT) (`  
  `void *prof);`

Parameter	Usage	Description
<code>prof</code>	Input	Pointer to the file being parsed.
Return Value	Description	
<code>&lt;item&gt;</code>	Item name, or NULL if there are no more items in the section.	

## GDB\_PROF\_GET\_VALUE

**Assignment** Attunity Connect specifies the name of this function as the `nv_prof_get_value` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function gets the value associated with a specified item in a section.

**Prototype** `char* (*GDB_FNC_PROF_GET_VALUE) (`  
  `void *prof,`  
  `char *section,`  
  `char *item);`

Parameter	Usage	Description
<code>prof</code>	Input	Pointer to the file being parsed.
<code>section</code>	Input	Name of section in which item is located.
<code>item</code>	Input	Name of item whose value you want to retrieve.

Return Value	Description
<code>&lt;value&gt;</code>	Value of item, or NULL if either the requested item does not exist in the section or the section does not exist.

## GDB\_PROF\_GET\_CUR\_VALUE

**Assignment** Attunity Connect specifies the name of this function as the **nv\_prof\_get\_cur\_value** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Description** This function gets the value associated with the current section and item.

**Prototype** `char* (*GDB_FNC_PROF_GET_CUR_VALUE) ( void *prof);`

Parameter	Usage	Description
<code>prof</code>	Input	Pointer to the file being parsed.
Return Value	Description	
<code>&lt;value&gt;</code>	Value of item, or NULL if either the current section or item is not set.	

## GDB\_PROF\_SET\_SECTION

**Assignment** Attunity Connect specifies the name of this function as the **nv\_prof\_set\_section** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Description** This function sets the current section to the specified section.

**Prototype** `char* (*GDB_FNC_PROF_SET_SECTION) ( void *prof, char *section);`

Parameter	Usage	Description
<code>prof</code>	Input	Pointer to the file being parsed.
<code>section</code>	Input	Name of section to be made current.
Return Value	Description	
<code>&lt;section_address&gt;</code>	Address of section, or NULL if the section is not found.	

## GDB\_PROF\_SET\_ITEM

**Assignment** Attunity Connect specifies the name of this function as the **nv\_prof\_set\_item** member of the GDB\_HELP\_FUNCTIONS structure. (See page 263.)

**Description** This function sets the current item to the specified item in the current section.

**Prototype** `char* (*GDB_FNC_PROF_SET_ITEM) (`  
  `void *prof,`  
  `char *item);`

Parameter	Usage	Description
prof	Input	Pointer to the file being parsed.
item	Input	Name of item to be made current.
Return Value	Description	
<item_address>	Address of item, or NULL if either the item is not found in the current section or the current section is not set.	

## GDB\_PROF\_DELETE

**Assignment** Attunity Connect specifies the name of this function as the `nv_prof_delete` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function frees all memory associated with this profile operation.

**Prototype** `(*GDB_FNC_PROF_DELETE) (void *prof);`

Parameter	Usage	Description
prof	Input	Pointer to the file being parsed.
Return Value	Description	
<status>	Upon successful return (TRUE_), the variable prof is no longer valid.	

## Database Property Functions

Some data sources allow you to set properties to modify the behavior of the data source in a particular instance. For example, you can set a “reserving clause” for an Rdb database to optimize performance when retrieving data from a read-only transaction, and later alter the reserving clause for a read-write transaction. Similarly, you may find that different property settings are required by a specific data source in a data driver. You can assign a property for a data source by:

- Calling the `DB_SET_PROPERTY` help function. For details, see page 184.
- Adding a `TDP_PROPERTIES` statement in the *data source name* section of the binding file for the data source definition.

- ❖ You can also set driver-specific properties in the DB\_COMMAND clause for the driver, but this mechanism does not enable you to turn a property setting on or off as needed by a particular data source.

Attunity Connect supports several properties specific to one or more of the supplied drivers, such as RESERVING\_CLAUSE (Rdb only) and DISABLE\_CURSORS (Sybase only). For details about valid properties for the supplied drivers, see the *Attunity Connect Reference*.

The following support functions enable you to define and check properties for a data source in a data driver:

- DB\_GET\_PROPERTY
- DB\_GET\_PROPERTY\_STRING
- DB\_GET\_PROPERTY\_LONG
- DB\_SET\_PROPERTY

## **DB\_GET\_PROPERTY**

**Assignment** Attunity Connect specifies the name of this function as the `nv_db_get_property` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function looks up the given property in the specified data source.

**Prototype** `void* (*GDB_FNC_DB_GET_PROPERTY) (`  
`char *tdp_name,`  
`char *property) ;`

Parameter	Usage	Description
<code>tdp_name</code>	Input	Name of the data source.
<code>property</code>	Input	Property to be checked in data source.

Return Value	Description
<code>&lt;value&gt;</code>	Value of property if found; otherwise returns NULL.

## **DB\_GET\_PROPERTY\_STRING**

**Assignment** Attunity Connect specifies the name of this function as the `nv_db_get_property_string` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function looks up the given property in the specified data source and returns the value as a string (regardless of the type used when the property was set).

**Prototype** `char* (*GDB_FNC_DB_GET_PROPERTY_STRING)(  
 char *tdp_name,  
 char *property);`

Parameter	Usage	Description
tdp_name	Input	Name of the data source.
property	Input	Property to be checked in data source.
Return Value	Description	
<value>	Value of property (as a string) if found; otherwise returns NULL.	

## DB\_GET\_PROPERTY\_LONG

**Assignment** Attunity Connect specifies the name of this function as the `nv_db_get_property_long` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function looks up the given property in the specified data source and returns the value as a long (regardless of the type used when the property was set).

**Prototype** `(*GDB_FNC_DB_GET_PROPERTY_LONG)(  
 char *tdp_name,  
 char *property);`

Parameter	Usage	Description
tdp_name	Input	Name of the data source.
property	Input	Property to be checked in data source.
Return Value	Description	
<value>	Value of property (as a long) if found; otherwise returns NULL.	

## DB\_SET\_PROPERTY

**Assignment** Attunity Connect specifies the name of this function as the `nv_db_set_property` member of the `GDB_HELP_FUNCTIONS` structure. (See page 263.)

**Description** This function sets (or adds) the given property in the data source, using the specified value and value type.

**Comments** You can also assign a property for a data source by adding a `TDP_PROPERTIES` statement in the *data source name* section of the binding file for the data source definition. The `TDP_PROPERTIES`

statement is more limited with regard to value type; all properties are set as string.

**Prototype** (\*GDB\_FNC\_DB\_SET\_PROPERTY) (

```
    char *tdp_name,
    char *property,
    void *value,
    long value_type);
```

Parameter	Usage	Description
tdp_name	Input	Name of the data source.
property	Input	Property to be set in data source.
value	Input	Value of property.
value_type	Input	Type of value to be set. Valid types are: VALUE_IS_LONG_ VALUE_IS_CHAR_ VALUE_IS_STRING_ The default is VALUE_IS_STRING_.
Return Value		Description
<status>		Successful return is TRUE_.



# Chapter 11 National Language Support

To map to and from a codepage not supported by Attunity Connect with a codepage supported by Attunity Connect, do the following:

- Define a text file that maps the required codepage to the supported codepage.
- Use the NAV\_UTIL CODEPAGE utility to generate a binary file with this mapping as follows:

```
nav_util CODEPAGE text_file
```

This generates a binary file named *<name of the language>.cp*, where *<name of the language>* is specified as the value for the NAME parameter in the text file whose syntax is described below.

The structure of the text file defining the mapping is as follows:

```
; This file defines the xxx language and related codepages for
; Attunity Connect
;
; To generate the xxx.cp file from this file use the following command
; (example for Windows):
;
; $ nav_util codepage xxx.txt
;
; The xxx.cp file is written to NAVROOT/def.
;
; [LANGUAGE] - Section defining general language properties
;
; NAME - The name of the defined language used in the binding properties
; the name of the generated .cp file in the NAVROOT/def directory.
; BASE - The base codepage for the language. All codepage tables
; defined in this file relate to conversions to/from this base codepage.
; Usually, the base codepage is selected based on several considerations:
;     - ASCII based (rather than EBCDIC based)
;     - Common in use
;     - Contain all required characters and symbols
; MS_WIN_CP - The Microsoft Windows codepage number to be used for this
; language. It must match the codepage used as the machine codepage in
; the environment definition.
;
; [LANGUAGE]
NAME = XXX
```

```
BASE = BASE_CODEPAGE
MS_WIN_CP = nnnn

; [CODEPAGES] - Section defining new codepages
;
; <codepage-name> = <conversion-table-name> - Each entry defines a new
; codepage and the conversion table to use for conversions from/to the
; base codepage. The tables mentioned here are defined later in this file.
;
[CODEPAGES]
CODEPAGE1 = CONVERSION1
CODEPAGE2 = CONVERSION2

; [DEFAULTS] - Section defining default codepages per platform
;
; <platform-name> = <codepage-name> - Each entry defines a default
; codepage for a platform. The last entry may specify the wildcard '*' as a
; platform to signify 'other platforms not specifically listed'
;
[DEFAULTS]
OS390 = CODEPAGE1
AS400 = CODEPAGE1
* = CODEPAGE2

; [XML-ENCODING] - Section defining encoding names for use in generated
; XML documents
;
; <xml-encoding-name> = <codepage-name> - Each entry defines an XML
; encoding name for a given codepage name. Thus, if a codepage is in effect,
; the XML encoding name used is determined in this section.
;
[XML-ENCODING]
ISO-8859-8 = CODEPAGE2
US-EBCDIC = CODEPAGE1

; [ALIAS] - Section defining codepage aliases.
;
; <alias-name> = <codepage-name> - Each entry of this form defines a
; codepage alias.
;
[ALIAS]
ISO-8859-8 = CODEPAGE2
ASCII = CODEPAGE2
US-EBCDIC = CODEPAGE1

; [<conversion-table-name>] - Sections with the name of a conversion
```

```

; table define the conversion tables used for converting values between
; the base codepage and the defined codepage. There are as many sets of
; conversion tables (two per codepage) as there are codepages to map.
;

; There are two tables defined per codepage:
; - The 'F' table (lines starting with Fi, i=0..F) describes the
;   conversion From the base codepage to the defined codepage.
; - The 'T' table (lines starting with Ti, i=0..F) describes the
;   conversion To the base codepage from the defined codepage.
;

; For example, consider the character 65 Dec (41 Hex) in the base
; codepage. To find its value in the defined codepage, look for the line
; starting with "F4 =" and find the 2nd value on the line (the first value
; refer to 40 Hex, the second to 41 Hex etc.).
; In a similar manner, to convert from the defined codepage to the base
; codepage, use the 'T' table.

; Conversion table
[CODEPAGE2]
;      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
F0 = "00 01 02 03 37 2D 2E 2F 16 05 15 0B 0C 0D 0E 0F"
F1 = "10 11 12 13 3C 3D 32 26 18 19 3F 27 22 1D 35 1F"
F2 = "40 5A 7F 7B 5B 6C 50 7D 4D 5D 5C 4E 6B 60 4B 61"
...
FE = "50 81 82 83 84 85 86 87 88 89 91 92 93 94 95 96"
FF = "97 98 99 A2 A3 A4 A5 A6 A7 A8 A9 FB B9 EA BB 41"

;      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
T0 = "00 01 02 03 EC 09 CA 7F E2 D2 D3 0B 0C 0D 0E 0F"
T1 = "10 11 12 13 EF 0A 08 CB 18 19 DC D8 00 1D FD 1F"
...
TE = "5C F6 53 54 55 56 57 58 59 5A FD F5 99 F7 F0 F9"
TF = "30 31 32 33 34 35 36 37 38 39 DB FB 9A F4 EA C9"

; [UNICODE] - Section defining the conversion between the base
; codepage and Unicode. Other codepages are converted to/from Unicode
; via the base codepage.
;

; Two tables are defined here:
; - The 'F' table (lines starting with Fi, i=0..F) describes the
;   conversion From the base codepage to Unicode.
; - The ranges table (lines starting with RNGi, i=0,1,...). Describes the
;   conversion to the base codepage from the Unicode. Each line has the
;   form:
;

; RNG<i> = <1st-Unicode-char> <last-Unicode-char> <start-codepage-char>
;
```

```
; For example, consider the character 65 Dec (41 Hex) in the base
; codepage. To find its Unicode value, we look for the line starting
; with "F4 =" and find the 2nd value on the line (the first value refer
; to 40 Hex, the second to 41 Hex etc.). The values are unsigned 16-bit
; numbers in hexadecimal.
;
; In a similar manner, to convert from a Unicode character C, scan the
; ranges table for a range that contains the C character. Then, get the
; codepage character value using this formula:
;
; <codepage-char> = <C> - <1st-Unicode-character> + <start-codepage-char>
;
[UNICODE]
;          0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
F0 = "0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 000A 000B 000C
000D 000E 000F"
F1 = "0010 0011 0012 0013 0014 0015 0016 0017 0018 0019 001A 001B 001C
001D 001E 001F"
F2 = "0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 002A 002B 002C
002D 002E 002F"
...
FF = "05E0 05E1 05E2 05E3 05E4 05E5 05E6 05E7 05E8 05E9 05EA 00FF 00FF
200E 200F 00ff"

;
RNG0 = "0000 00A9 00"
RNG1 = "05D0 05EA E0"
...
RNG5 = "200E 200F FD"
```

**Part A**

## **Appendix – Data Driver Example**



# Appendix A Sample Data Drivers

## Skeleton Data Driver

This sample data driver “skeleton” shows the minimum set of tasks that each data driver needs to perform. These tasks include setting up and allocating memory, as well as standard coding techniques such as returning a function’s status. The driver skeleton also illustrates the use of several predefined support routines to perform low-level functions and simplify the driver code. (For details about these routines, see “Developer SDK Predefined Functions” on page 153.)

- ❖ The driver skeleton does not supply any metadata routines (GET\_ITEM\_LIST, DESCRIBE\_TABLE, and so on), because the LOAD function sets up the driver to work with Attunity Connect Data Dictionary (ADD).

This section describes the following topics:

- Setup
- Connect
- Disconnect
- Set Buffer
- Open
- Set Index
- Fetch
- Close
- Rewind
- Get Last Error
- Load
- The Driver Skeleton Metadata (.adl)
- The Driver Skeleton Binding File Entry
- The Driver Skeleton ADDON.DEF Entry

## Setup

The setup code includes the GDB header file **dbgdb.h** (described on page 19).

The setup code also establishes a pointer to the predefined support functions in the structure GDB\_HELP\_FUNCTIONS, using the macro GDB\_HELP\_DEFINE. (See "Developer SDK Predefined Functions" on page 153.)

**Code Skeleton:**

```
#include "dbgdb.h"

#ifndef WIN32
#define EXPORT_SYMBOL __declspec(dllexport)
#else
#define EXPORT_SYMBOL
#endif

GDB_HELP_DEFINE ;

/* Connection context structure - used as the connection handle */
typedef struct {
    void          *memory ;
    STRING        last_error_text ;
    /* ... */
} XXX_CONNECT ;

/* Stream context structure - used as the stream handle */
typedef struct {
    void          *memory ;
    TABLE_DA     *table_da ;
    XXX_CONNECT *xxx_connect ;
    void          *buffer_data ;
    long          is_index_set ;
    /* ... */
} XXX_STREAM ;
```

## Connect

**Code Sample:**

```
=====
 * Connect to a xxx database
=====

static GDB_STATUS db_xxx_connect(
    GDB_HANDLE *gdb_handle,
    char *application_name,
    char *connect_string,
    char *username_password)
```

```
{
    XXX_CONNECT *xxx_connect ;
    void *memory ;
    /* Allocate the XXX_CONNECT structure for this connection */
    memory = MEMP_NEW("XXX CONNECT", 0) ;
    xxx_connect = MEMP_MALLOC(sizeof(XXX_CONNECT), memory) ;
    xxx_connect->memory = memory ;
    /* The XXX_CONNECT structure is used as the connection handle */
    *gdb_handle = xxx_connect ;
    /* Prepare dynamic strings for the error text */
    DSTR_NEW(&xxx_connect->last_error_text, "", 0, 256, memory) ;
    return (GDB_OK_) ;
}
```

## Disconnect

### Code Sample:

```
/*=====
 * Disconnect from a xxx database
 *=====
static GDB_STATUS db_xxx_disconnect(GDB_HANDLE gdb_handle)
{
    XXX_CONNECT *xxx_connect = gdb_handle ;
    void *memory = xxx_connect->memory ;
    /* Free the connection memory pool */
    MEMP_DELETE(memory) ;
    return (GDB_OK_) ;
}
```

## Open

### Code Sample:

```
/*=====
 * Open a stream on a table
 *=====
static GDB_STATUS db_xxx_open(
    GDB_STREAM *io_stream,
    GDB_HANDLE gdb_handle,
    TABLE_DA *table_da,
    GDB_OPEN_MODE mode)
{
    XXX_CONNECT *xxx_connect = gdb_handle ;
    XXX_STREAM *xxx_stream ;
    void *memory ;
```

```
    memory = MEMP_NEW("XXX_STREAM", 0) ;
    xxx_stream = MEMP_MALLOC(sizeof(XXX_STREAM), memory) ;
    xxx_stream->memory = memory ;
    xxx_stream->xxx_connect = xxx_connect ;
    xxx_stream->table_da = table_da ;
    xxx_stream->is_index_set = FALSE ;
    xxx_stream->first_fetch = TRUE ;
    /* Return the xxx_stream structure as the stream handle */
    *io_stream = xxx_stream ;
    return (GDB_OK_) ;
}
```

## Set Buffer

### Code Sample:

```
/*=====
 * Set output buffer for the stream
 *=====
static GDB_STATUS db_xxx_set_buffer(GDB_STREAM io_stream,
                                    GDB_BUFFER *io_buffer)
{
    XXX_STREAM *xxx_stream = io_stream ;
    XXX_CONNECT *xxx_connect = xxx_stream->xxx_connect ;
    TABLE_DA *table_da = xxx_stream->table_da ;
    void *memory = xxx_stream->memory ;
    /* This is the address of the buffer that fetch must populate */
    xxx_stream->buffer_data = io_buffer->buffer_data ;
    return (GDB_OK_) ;
}
```

## Set Index

### Code Sample:

```
/*=====
 * Set to a specific index
 *=====
static GDB_STATUS db_xxx_set_index(
    GDB_STREAM io_stream,
    long index_number,
    GDB_INDEX_RELATION index_relation,
    GDB_BUFFER *start_buffer,
    GDB_BUFFER *end_buffer)
{
    XXX_STREAM *xxx_stream = io_stream ;
```

```

XXX_CONNECT *xxx_connect = xxx_stream->xxx_connect ;
TABLE_DA *table_da = xxx_stream->table_da ;
void *memory = xxx_stream->memory ;
xxx_stream->is_index_set = TRUE ;
xxx_stream->first_fetch = TRUE ;

/* Call to backend set-index-range API */

return (GDB_OK_) ;
}

```

## Fetch

### Code Sample:

```

/*=====
 * Fetch one object from the class
 *=====
static GDB_STATUS db_xxx_fetch(GDB_STREAM io_stream, GDB_BOOKMARK bm)
{
    XXX_STREAM *xxx_stream = io_stream ;
    XXX_CONNECT *xxx_connect = xxx_stream->xxx_connect ;
    TABLE_DA *table_da = xxx_stream->table_da ;
    void *memory = xxx_stream->memory ;
    if (xxx_stream->first_fetch && !xxx_stream->is_index_set) {
        /* Use whatever index to scan sequentially */
    }
    xxx_stream->first_fetch = FALSE ;
    xxx_stream->is_index_set = FALSE ;

    /* call to backend fetch API into xxx_stream->buffer_data */

    return (GDB_OK_) ;
}

```

## Close

### Code Sample:

```

/*=====
 * Close a stream
 *=====
static GDB_STATUS db_xxx_close(GDB_STREAM io_stream)
{
    XXX_STREAM *xxx_stream = io_stream ;
    XXX_CONNECT *xxx_connect = xxx_stream->xxx_connect ;

```

```
    TABLE_DA *table_da = xxx_stream->table_da ;
    void *memory = xxx_stream->memory ;
    GDB_STATUS status ;
    /* call to backend close API */
    MEMP_DELETE(memory) ;
    return (GDB_OK_) ;
}
```

## Rewind

### Code Sample:

```
/*=====
 * Rewind a stream
 *=====
static GDB_STATUS db_xxx_rewind(GDB_STREAM io_stream)
{
    XXX_STREAM *xxx_stream = io_stream ;
    XXX_CONNECT *xxx_connect = xxx_stream->xxx_connect ;
    TABLE_DA *table_da = xxx_stream->table_da ;
    void *memory = xxx_stream->memory ;
    return (GDB_OK_) ;
}
```

## Get Last Error

### Code Sample:

```
/*=====
 * Get the error text for the last error
 *=====
static char *db_xxx_get_last_error(GDB_HANDLE gdb_handle)
{
    XXX_CONNECT *xxx_connect = gdb_handle ;
    if (!xxx_connect)
        return ("") ;
    return (xxx_connect->last_error_text) ;
}
```

## Load

The LOAD function populates the "GDB\_DB\_FUNCTIONS" structure to identify the routines that you are supplying for the driver.

Note the following special requirements for the external LOAD function (`db_xxx_get_functions`):

- The LOAD function is written in such a way that it can be exposed by the DLL. Typically it is the only symbol that the driver needs to expose.
- The `GDB_HELP_FUNCTIONS` structure is passed to the driver's LOAD function. (In earlier versions of the GDB API, only the `GDB_DB_FUNCTIONS` structure was passed.)
- The LOAD function initializes the `GDB_HELP_FUNCTIONS` structure and the version member of the function pointer, using the following macro statement:

```
GDB_INITIALIZE(fnc, help);
```

where:

`fnc` – pointer to "`GDB_DB_FUNCTIONS`" structure containing function definitions for a data driver.

`help` – "`GDB_HELP_FUNCTIONS`" argument passed to the entry point function (INIT-FUNCTION) in the DLL.

### Code Sample

```
/*=====
 * Get the driver function vector
 *=====
BASE_EXPORT GDB_STATUS db_xxx_get_functions(GDB_DB_FUNCTIONS *fnc,
                                             GDB_HELP_FUNCTIONS *help)
{
    GDB_INITIALIZE(fnc, help) ;
    fnc->io_connect = db_xxx_connect ;
    fnc->io_disconnect = db_xxx_disconnect ;
    fnc->io_open = db_xxx_open ;
    fnc->io_set_buffer = db_xxx_set_buffer ;
    fnc->io_set_index = db_xxx_set_index ;
    fnc->io_fetch = db_xxx_fetch ;
    fnc->io_close = db_xxx_close ;
    fnc->io_rewind = db_xxx_rewind ;
    fnc->io_get_last_error = db_xxx_get_last_error ;
    GDB_SET_ADD(fnc) ;
    /* Set to "real" bookmark size when adding update capability */
    GDB_SET_BOOKMARK_SIZE(fnc, 0) ;
    return (GDB_OK_) ;
}
```

## The Driver Skeleton Metadata (.adl)

The following ADDL code defines a simple table XXXTABLE for the driver skeleton:

```
DEFINE RECORD XXXTABLE.  
  FIELDS.  
    COL1  
      DATATYPE IS STRING  
      SIZE IS 25 CHARACTERS.  
  END FIELDS.  
END RECORD.
```

When the adl file is compiled into a binary rec file, the metadata is stored in Attunity Connect Data Dictionary (ADD).

## The Driver Skeleton ADDON.DEF Entry

To specify this sample driver skeleton in the Attunity Connect environment, create a custom define file dbxxx.def as follows:

```
[DB_XXX_GDB]  
TYPE=DRIVER  
INIT-FUNCTION=db_xxx_get_functions  
SHAREABLE-NAME=NVDB_XXX
```

Merge this definition with any pre-existing definitions in the addon.def file, using the following command:

```
nav_util addon dbxxx.def
```

Prior to running Attunity Connect with this driver, you must make sure that Attunity Connect can access the DLL that contains the driver code, identified as NVDB\_XXX in addon.def. See "System Notes" on page 253 for platform-specific information about this task.

## The Driver Skeleton Binding File Entry

To specify a data source based on this sample driver, edit the binding file as follows:

```
[TDP-NAMES]  
XXX_GDB = DB_XXX_GDB  
  
...
```

```
[XXX_GDB]
TDP_CONNECT = d:\src\gdb,connectinfo
```

where:

**XXX\_GDB** – The name used to identify the data source to Attunity Connect and which applications use to access the data.

**DB\_XXX\_GDB** – The data driver name that you specify when you register the driver in the addon.def file.

**d:\src\gdb,connectinfo** – The specific information Attunity Connect needs to connect to this data source. Since this driver skeleton uses metadata from Attunity Connect Data Dictionary (ADD), the first part of the connect string identifies the ADD directory, followed by a comma and the actual connect information required in order to access the data source. The connection information is supplied as input to the CONNECT function (see page 194).

## DISAM-Like Data Driver

The following driver shows how to access data similar to DISAM, referred to as XISAM in the code on a Windows platform (which is not alignment sensitive). It can be modified to access other data sources that are similar to DISAM or CISAM.

```
/*
 * Copyright (c) 2001, by Attunity Ltd. ALL RIGHTS RESERVED.
 */

#include <stdio.h>

#include <xisam.h>
#include <xisamdec.h>

#include "dbgdb.h"
#include "dtypeid.h"

#ifndef WIN32
#include <windows.h>
#define BASE_EXPORT __declspec(dllexport)
#else
#define BASE_EXPORT
#endif

GDB_HELP_DEFINE;
```

```
#define BOOKMARK_SIZE  (sizeof(long)) /*recnum is long */
#define LONG_SIZE 4
#define SHORT_SIZE 2

/* Connection context structure - used as the connection handle */
typedef struct {
    void          *memory;
    char          *last_error_text;
} XISAM_CONNECT;

/* Stream context structure - used as the stream handle */
typedef struct {
    void          *memory;
    TABLE_DA      *table_da;
    XISAM_CONNECT*xisam_connect;
    void          *buffer_data;
    void          *key_data;
    long           is_index_set;
    long           first_fetch;
    long           fd;
    long           rec_size;
    long           read_mode;
    GDB_BOOKMARK bm;
    GDB_DYNARRAY index_array;
} XISAM_STREAM;

#define STR_COPY(trg,src)
    memcpy(trg, src, min(strlen(src), MAX_FULLNAME_LENGTH_)+1);

#define STR_APPEND(trg,src)
    strcpy((trg)+strlen(trg), src);

#define GET_BOOKMARK_VAL(bm)  (*((long*) (bm)))
static void set_error(XISAM_CONNECT *fconnect, long id, char *text, char *where);
static GDB_STATUS read_record(XISAM_STREAM *xisam_stream);
static GDB_STATUS store_key(struct keydesc *key, XISAM_STREAM
*xisam_stream, GDB_BUFFER *start_buffer);
static GDB_STATUS load_buffer(XISAM_STREAM *xisam_stream);
static GDB_STATUS store_buffer(XISAM_STREAM *xisam_stream);
static GDB_STATUS set_key(XISAM_CONNECT *diasm_connect, TABLE_DA
*table_da,
                           INDEX_DA *index_da, struct keydesc
*key);
static GDB_STATUS create_key_array(XISAM_STREAM *xisam_stream, TABLE_DA
*table_da);
static GDB_STATUS create_new_table(XISAM_STREAM *xisam_stream, TABLE_DA
```

```
*table_da);
static GDB_STATUS compute_recsize(long *recsize, TABLE_DA *table_da);

/*===== G D B      A P I      F U N C T I O N S =====*/
/*=====
 * Connect to an xisam database
 *=====
static GDB_STATUS db_xisam_connect(GDB_HANDLE *gdb_handle,
        char *application_name,
        char *connect_string,
        char *username_password)
{
    XISAM_CONNECT *xisam_connect;
    void *memory;

    /* Allocate the XISAM_CONNECT structure for this connection */
    memory = MEMP_NEW("XISAM CONNECT", 0);
    xisam_connect = MEMP_MALLOC(sizeof(XISAM_CONNECT), memory);
    xisam_connect->memory = memory;

    /* Set error text */
    xisam_connect->last_error_text = STR_C_STRING("", 500, memory);

    /* The XISAM_CONNECT structure is used as the connection handle */
    *gdb_handle = xisam_connect;

    return (GDB_OK_);
}

/*=====
 * Disconnect from database
 *=====
static GDB_STATUS db_xisam_disconnect(GDB_HANDLE gdb_handle)
{
    XISAM_CONNECT *xisam_connect = gdb_handle;
    void *memory = xisam_connect->memory;

    /* Free the connection memory pool */
    MEMP_DELETE(memory);

    return (GDB_OK_);
}

/*=====
```

```
* Open a stream on a table
*=====
static GDB_STATUS db_xisam_open(
    GDB_STREAM *io_stream,
    GDB_HANDLE gdb_handle,
    TABLE_DA *table_da,
    GDB_OPEN_MODE mode)
{
    XISAM_CONNECT *xisam_connect = gdb_handle;
    XISAM_STREAM *xisam_stream;
    void *memory;
    struct dictinfo info;
    int* error;
    long ismode;
    GDB_STATUS status;

    memory = MEMP_NEW("XISAM STREAM", 0);
    xisam_stream = MEMP_MALLOC(sizeof(XISAM_STREAM), memory);
    xisam_stream->memory = memory;
    xisam_stream->xisam_connect = xisam_connect;
    xisam_stream->table_da = table_da;
    xisam_stream->bm = NULL;
    xisam_stream->is_index_set = FALSE;
    xisam_stream->first_fetch = TRUE;

    /* determine open mode */
    if (mode == GDB_OPEN_READ_ONLY_) {
        ismode = ISINPUT | ISMANULOCK;
    }
    else {
        ismode = ISINOUT | ISMANULOCK;
    }

    /* check path of table */
    if (table_da->path == NULL) {
        set_error(xisam_connect, 0, "Table path is not defined.",
"OPEN");
        return (GDB_NOT_OK_);
    }

    /* get index info - do before create_new_table*/
    status = create_key_array(xisam_stream, table_da);
    if (status != GDB_OK_) return status;

    /* try to open table */
    if ((xisam_stream->fd = isopen(table_da->path, ismode)) < 0) {
```

```
/* Creating the table is postponed till here to give the user
a chance to create a primary index */
    if (mode == GDB_OPEN_READ_WRITE_) { /* This is not the
greatest way to detect this situation, would be better to see if the .dat
file was there.*/

        status = create_new_table(xisam_stream, table_da);
        if (status != GDB_OK_) return status;

        /* try to open again */
        xisam_stream->fd = isopen(table_da->path, ismode);
    }

    if (xisam_stream->fd < 0) {
        error = is_errno(xisam_stream->fd);
        set_error(xisam_connect, *error, "Error opening xisam
table.", "OPEN");
        return (GDB_NOT_OK_);
    }
}

/* Get record size */
isindexinfo(xisam_stream->fd, (void*)&info, 0);
xisam_stream->rec_size = info.di_recsize;

/* Return the xisam_stream structure as the stream handle */
*io_stream = xisam_stream;

return (GDB_OK_);
}

/*=====
 * Set output buffer for the stream
 *=====
 */

static GDB_STATUS db_xisam_set_buffer(GDB_STREAM io_stream,
                                      GDB_BUFFER *io_buffer)
{
    XISAM_STREAM *xisam_stream = io_stream;
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    TABLE_DA *table_da = xisam_stream->table_da;
    void *memory = xisam_stream->memory;

    /* This is the address of the buffer that fetch must populate */
    xisam_stream->buffer_data = io_buffer->buffer_data;
```

```
        return (GDB_OK_);  
    }  
  
/*===== * Set to a specific index =====*/  
  
static GDB_STATUS db_xisam_set_index(  
    GDB_STREAM io_stream,  
    long index_number,  
    GDB_INDEX_RELATION index_relation,  
    GDB_BUFFER *start_buffer,  
    GDB_BUFFER *end_buffer)  
{  
    XISAM_STREAM *xisam_stream = io_stream;  
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;  
    TABLE_DA *table_da = xisam_stream->table_da;  
    void *memory = xisam_stream->memory;  
    int read_mode;  
    GDB_STATUS status;  
    int* error;  
    struct keydesc *key;  
  
    xisam_stream->is_index_set = TRUE;  
    xisam_stream->first_fetch = TRUE;  
  
    /* Set the read_mode */  
    if (index_relation == GDB_EQUAL_) {  
        read_mode = ISEQUAL;  
    }  
    else { /* GDB_GREATER_EQUAL_ */  
        read_mode = ISGTEQ;  
    }  
  
    /* get the key structure */  
    key = xisam_stream->index_array[index_number];  
  
    /* store key */  
    status = store_key(key, xisam_stream, start_buffer);  
    if (status != GDB_OK_) return status;  
  
    if (isstart(xisam_stream->fd, key, start_buffer->buffer_length,  
    xisam_stream->key_data, read_mode) < 0) {  
        /* check error */  
        error = is_errno(xisam_stream->fd);  
        if (*error != ENOREC) {
```

```
        set_error(xisam_connect, *error, "Error setting
index.", "SETINDEX");
                return (GDB_NOT_OK_);
        }
        return (GDB_NOT_FOUND_); /* no records found */
    }

xisam_stream->bm = NULL; /* current record has been set by isstart
*/
xisam_stream->read_mode = ISCURR; /* next read will be ISCURR */

return (GDB_OK_);
}

/*=====
 * Fetch one object from the class
 *=====
static GDB_STATUS db_xisam_fetch(GDB_STREAM io_stream, GDB_BOOKMARK bm)
{
    XISAM_STREAM *xisam_stream = io_stream;
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    TABLE_DA *table_da = xisam_stream->table_da;
    void *memory = xisam_stream->memory;
    struct keydesc key;
    long *recnum;
    GDB_STATUS status;

    if (xisam_stream->first_fetch && !xisam_stream->is_index_set &&
        !xisam_stream->bm) {

        /* Use no index to scan sequentially */
        key.k_nparts = 0;
        if (isstart(xisam_stream->fd, &key, 0,
xisam_stream->buffer_data, ISEQUAL) < 0) {
            set_error(xisam_connect, 0, "Error setting index.",
"FETCH");
            return (GDB_NOT_OK_);
        }

        /* set the record pos */
        xisam_stream->bm = NULL; /* current record set by isstart */
        xisam_stream->read_mode = ISCURR;
    }
    else
        if(xisam_stream->fd)
    {

```

```
        long num_bm;
        int test_num;
        num_bm = *(long*)bm;
        test_num = isgoto(xisam_stream->fd, num_bm);
    }

xisam_stream->first_fetch = FALSE;
xisam_stream->is_index_set = FALSE;

/* read record */
status = read_record(xisam_stream);
if (status != GDB_OK_) return status;

xisam_stream->read_mode = ISNEXT; /* the next read will be ISNEXT */

status = load_buffer(xisam_stream);

/* get bookmark */
if (bm != NULL) { /* may be null */
    recnum = is_recnum(xisam_stream->fd);
    memcpy(bm, recnum, BOOKMARK_SIZE);
}

return (status);
}

/*=====
 * Close a stream
 *=====
 */

static GDB_STATUS db_xisam_close(GDB_STREAM io_stream)
{
    XISAM_STREAM *xisam_stream = io_stream;
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    TABLE_DA *table_da = xisam_stream->table_da;
    void *memory = xisam_stream->memory;
    GDB_STATUS status = GDB_OK_;
    int* error;

    /* call to backend close API */
    if (isclose(xisam_stream->fd) < 0) {
        error = is_errno(xisam_stream->fd);
        set_error(xisam_connect, *error, "Error closing.", "CLOSE");
        status = GDB_NOT_OK_;
    }

    /* clean up memory */
}
```

```
    MEMP_DELETE(memory);

    return (status);
}

/*=====
 * Rewind a stream
 *=====
 */
static GDB_STATUS db_xisam_rewind(GDB_STREAM io_stream)
{
    XISAM_STREAM *xisam_stream = io_stream;

    xisam_stream->first_fetch = TRUE;

    return (GDB_OK_);
}

/*=====
 * Get the error text for the last error
 *=====
 */
static char *db_xisam_get_last_error(GDB_HANDLE gdb_handle)
{
    XISAM_CONNECT *xisam_connect = gdb_handle;

    if (!xisam_connect)
        return ("");

    return (xisam_connect->last_error_text);
}

/*=====
 * Set the bookmark
 *=====
 */
static GDB_STATUS db_xisam_set_bm(GDB_STREAM io_stream,
                                  GDB_BOOKMARK bm,
                                  GDB_LOCK_MODE lock_mode)
{
    XISAM_STREAM *xisam_stream = io_stream;

    /* set bookmark and ISCURR so read_record will set position */
    xisam_stream->bm = bm;
    xisam_stream->read_mode = ISCURR;

    if (lock_mode == GDB_LOCK_) {
```

```
        xisam_stream->read_mode |= ISLOCK; /* next read will be
ISCURR and locked*/
    }

    return (GDB_OK_);
}

/*=====
 * Convert the bookmark data
 *=====
static GDB_STATUS db_xisam_convert_bm_data(GDB_STREAM io_stream,
GDB_BM_CONVERT_MODE mode,
GDB_BOOKMARK bm,
char *str_bm)
{
    long lval;

    switch (mode) {

        case GDB_BM_FROM_STRING_:
            lval = atoi(str_bm);
            memcpy((void *)bm, (void *)&lval,sizeof(long));
            break;

        case GDB_BM_TO_STRING_:
            memcpy((void *)&lval, (void *)bm,sizeof(long));
            sprintf(str_bm, "%d", lval);
            break;
    }

    return (GDB_OK_);
}

/*=====
 * Delete the row
 *=====
static GDB_STATUS db_xisam_delete_row(GDB_STREAM io_stream, GDB_BOOKMARK
bm)
{
    XISAM_STREAM *xisam_stream = io_stream;
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    long* recnum;
    int* error;

    /* set the current record */
```

```
recnum = is_recnum(xisam_stream->fd);
*recnum = GET_BOOKMARK_VAL(bm);

/* delete the record */
if (isdelcurr(xisam_stream->fd) < 0) {
    error = is_errno(xisam_stream->fd);
    set_error(xisam_connect, *error, "Error deleting record.", "DELETE");
    return (GDB_NOT_OK_);
}

/* force the write */
if (isflush(xisam_stream->fd) < 0) {
    error = is_errno(xisam_stream->fd);
    set_error(xisam_connect, *error, "Error deleting record.", "DELETE - FLUSH");
    return (GDB_NOT_OK_);
}

return (GDB_OK_);
}

/*=====
 * Add a row
 *=====
 */
static GDB_STATUS db_xisam_add_row(GDB_STREAM io_stream, GDB_BOOKMARK bm)
{
    XISAM_STREAM *xisam_stream = io_stream;
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    GDB_STATUS status;
    int* error;
    long* recnum;

    /* convert the data */
    status = store_buffer(xisam_stream);
    if (status != GDB_OK_) return status;

    /* write it out */
    if (iswrite(xisam_stream->fd, xisam_stream->buffer_data) < 0) {
        error = is_errno(xisam_stream->fd);
        if (*error == EDUPL) {
            return (GDB_DUPLICATE_KEY_IN_INDEX_);
        }
        else {
            if (*error == ELOCKED) {
                return (GDB_RESOURCE_LOCKED_);
            }
        }
    }
}
```

```
        }
    else { /* general error */
        set_error(xisam_connect, *error, "Error adding
record.", "ADD");
        return (GDB_NOT_OK_);
    }
}

/* force the write */
if (isflush(xisam_stream->fd) < 0) {
    error = is_errno(xisam_stream->fd);
    set_error(xisam_connect, *error, "Error adding record.", "ADD
- FLUSH");
    return (GDB_NOT_OK_);
}

/* get bookmark */
if (bm != NULL) { /* may be null */
    recnum = is_recnum(xisam_stream->fd);
    memcpy(bm, recnum, BOOKMARK_SIZE);
}

return (GDB_OK_);
}

/*=====
 * Update a row
 *=====
 */

static GDB_STATUS db_xisam_update_row(GDB_STREAM io_stream, GDB_BOOKMARK
bm)
{
    XISAM_STREAM *xisam_stream = io_stream;
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    GDB_STATUS status;
    int* error;

    /* convert the data */
    status = store_buffer(xisam_stream);
    if (status != GDB_OK_) return status;

    /* update the record */
    if (isrewrec(xisam_stream->fd, GET_BOOKMARK_VAL(bm),
xisam_stream->buffer_data) < 0) {
        error = is_errno(xisam_stream->fd);
```

```

        if (*error == EDUPL) {
            return (GDB_DUPLICATE_KEY_IN_INDEX_);
        }
        else {
            if (*error == ELOCKED) {
                return (GDB_RESOURCE_LOCKED_);
            }
            else { /* general error */
                set_error(xisam_connect, *error, "Error updating
record.", "UPDATE");
                return (GDB_NOT_OK_);
            }
        }
    }

/* force the write */
if (isflush(xisam_stream->fd) < 0) {
    error = is_errno(xisam_stream->fd);
    set_error(xisam_connect, *error, "Error updating record.",
"UPDATE - FLUSH");
    return (GDB_NOT_OK_);
}

return (GDB_OK_);
}

/*=====
 * Lock a row
 *=====
 */
static GDB_STATUS db_xisam_lock_row(GDB_STREAM io_stream, GDB_BOOKMARK
bm)
{
    XISAM_STREAM *xisam_stream = io_stream;

    xisam_stream->bm = bm; /* set bookmark for current read */
    xisam_stream->read_mode = ISCURR | ISLOCK;

    /* need to call read in xisam to lock row */
    return read_record(xisam_stream);

}

/*=====
 * Unlock a row
 *=====
 */

```

```
static GDB_STATUS db_xisam_unlock_row(GDB_STREAM io_stream, GDB_BOOKMARK
bm)
{
    XISAM_STREAM *xisam_stream = io_stream;
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    int* error;

    if (isrelrec(xisam_stream->fd, GET_BOOKMARK_VAL(bm)) < 0) {
        error = is_errno(xisam_stream->fd);
        set_error(xisam_connect, *error, "Error unlocking row.",
"UNLOCK_ROW");
        return (GDB_NOT_OK_);
    }

    return (GDB_OK_);
}

/*=====
 * Create a table
 *=====
*/
static GDB_STATUS db_xisam_create_table(GDB_HANDLE gdb_handle,
    TABLE_DA *table_da)
{
    /* Do to the restrictions of XISAM where the primary index needs to
be defined at the time of table creation the actual creating is done on
the first insert in the open function.

    This can be accomplished with the ADD support the GDB provides.
Since this driver is marked as ADD, the metadata needed to create the
table and indexes is stored in the rec file and returned during the open
in TABLE_DA.

    Thus this function is really a no-op and does not need to be defined.
 */

    return (GDB_OK_);
}

/*=====
 * Create an index
 *=====
*/
static GDB_STATUS db_xisam_create_index(GDB_HANDLE gdb_handle,
    TABLE_DA *table_da,
    long index_number)
```

```
{  
    XISAM_CONNECT *xisam_connect = (XISAM_CONNECT*)gdb_handle;  
    long fd;  
    struct keydesc key;  
    GDB_STATUS status;  
    int *error;  
  
    /* If the table is valid then add an index to it, otherwise let the  
    ADD support handle it and create the index at table creation. The primary  
    index would be a good example of this. */  
  
    /* check to see if the file is there */  
    if ((fd = isopen(table_da->path, ISEXCLLOCK+ISINOUT)) < 0) {  
        return (GDB_OK_); /* assuming the table hasn't been created  
yet, it would be better to check if the file existed */  
    }  
  
    /* set key structure */  
    status = set_key(xisam_connect, table_da,  
&table_da->indexs[index_number], &key);  
    if (status != GDB_OK_) return status;  
  
    /* add the index */  
    if (isaddindex(fd, &key) < 0) {  
        error = is_errno(fd);  
        set_error(xisam_connect, *error, "Error creating index.",  
"CREATE_INDEX");  
        return (GDB_NOT_OK_);  
    }  
  
    return (GDB_OK_);  
}  
  
/*=====*  
 * Drop a table  
 *=====*/  
  
static GDB_STATUS db_xisam_drop_table(GDB_HANDLE gdb_handle,  
    TABLE_DA *table_da)  
{  
    XISAM_CONNECT *xisam_connect = (XISAM_CONNECT*)gdb_handle;  
  
    if (iserase(table_da->path) < 0) {  
        set_error(xisam_connect, 0, "Error removing table.",  
"DROP_TABLE");  
        return (GDB_NOT_OK_);  
    }
```

```
        return (GDB_OK_);  
    }  
  
/*===== * Get the driver function vector =====*/  
  
BASE_EXPORT GDB_STATUS db_xisam_get_functions(GDB_DB_FUNCTIONS *fnc,  
                                              GDB_HELP_FUNCTIONS *help)  
{  
    GDB_INITIALIZE(fnc, help);  
  
    fnc->io_connect = db_xisam_connect;  
    fnc->io_disconnect = db_xisam_disconnect;  
    fnc->io_open = db_xisam_open;  
    fnc->io_set_buffer = db_xisam_set_buffer;  
    fnc->io_set_index = db_xisam_set_index;  
    fnc->io_fetch = db_xisam_fetch;  
    fnc->io_close = db_xisam_close;  
    fnc->io_rewind = db_xisam_rewind;  
    fnc->io_get_last_error = db_xisam_get_last_error;  
    fnc->io_set_bm = db_xisam_set_bm;  
    fnc->io_add_row = db_xisam_add_row;  
    fnc->io_update_row = db_xisam_update_row;  
    fnc->io_delete_row = db_xisam_delete_row;  
    fnc->io_lock_row = db_xisam_lock_row;  
    fnc->io_unlock_row = db_xisam_unlock_row;  
    fnc->io_drop_table = db_xisam_drop_table;  
    fnc->io_create_table = db_xisam_create_table;  
    fnc->io_create_index = db_xisam_create_index;  
    fnc->io_convert_bm_data = db_xisam_convert_bm_data;  
  
    GDB_SET_ADD(fnc);  
    GDB_SET_SUPPORT_UPDATE(fnc);  
    GDB_SET_BOOKMARK_SIZE(fnc, BOOKMARK_SIZE);  
  
    return (GDB_OK_);  
}  
  
/*===== * Write a formatted error message to last_error_text =====*/  
  
static void set_error(XISAM_CONNECT *xisam_connect,  
                      long id, char *text, char *where)  
{
```

```
        sprintf(xisam_connect->last_error_text,
                  "XISAMGDB (%d): %s; %s", id, text, where);
    }

/*=====
 *
 * Reads a record
 *=====
 */

static GDB_STATUS read_record(XISAM_STREAM *xisam_stream)
{
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    int* error;

    /* check to see if there is a bookmark set and read mode is ISCURR */
    /*if ((xisam_stream->bm != NULL) && (xisam_stream->read_mode &
    ISCURR)) {
        recnum = is_recnum(xisam_stream->fd);
        *recnum = GET_BOOKMARK_VAL(xisam_stream->bm);      /* set the
current record based on bookmark */
    }*/

    /* read the data in */
    if (isread(xisam_stream->fd, xisam_stream->buffer_data,
    xisam_stream->read_mode) < 0) {
        /* check error */
        error = is_errno(xisam_stream->fd);
        if (*error == EENDFILE) {
            return (GDB_END_OF_STREAM_);
        }

        /* general error */
        set_error(xisam_connect, *error, "Error reading record.",
    "FETCH");
        return (GDB_NOT_OK_);
    }

    return (GDB_OK_);
}

/*=====
 *
 * Converts the key data buffer
 *=====
 */

static GDB_STATUS store_key(struct keydesc *key,
    XISAM_STREAM *xisam_stream,
    GDB_BUFFER *start_buffer)
```

```
{  
  
    struct keypart *keyp;  
    int part;  
    long lval;  
    short ival;  
  
    /* create buffer if needed */  
    if (xisam_stream->key_data == NULL) {  
        xisam_stream->key_data = MEMP_MALLOC(xisam_stream->rec_size,  
xisam_stream->memory);  
    }  
  
    /* copy key */  
    memcpy(xisam_stream->key_data, start_buffer->buffer_data,  
start_buffer->buffer_length);  
  
    for (part = 0; part < key->k_nparts; part++) {  
        keyp = &key->k_part[part];  
  
        switch (keyp->kp_type) {  
  
            case LONGTYPE:  
                memcpy(&lval,  
(void*)((char*)start_buffer->buffer_data + keyp->kp_start), LONG_SIZE);  
                stlong(lval,  
(void*)((char*)xisam_stream->key_data + keyp->kp_start));  
                break;  
  
            case INTTYPE:  
                memcpy(&ival,  
(void*)((char*)start_buffer->buffer_data + keyp->kp_start), SHORT_SIZE);  
                stint(ival,  
(void*)((char*)xisam_stream->key_data + keyp->kp_start));  
                break;  
  
            default:  
                break;  
        }  
    }  
  
    return GDB_OK_;  
}  
  
/*===== * Converts the incoming buffer =====*/
```

```
static GDB_STATUS load_buffer(XISAM_STREAM *xisam_stream)
{
    TABLE_DA *table_da = xisam_stream->table_da;
    COLUMN_DA *column_da;
    long offset = 0;
    long lval, col;
    short ival;
    long filter_length;

    for (col = 0; col < table_da->n_columns; col++) {

        column_da = &table_da->columns[col];

        /* check type */
        switch (column_da->datatype) {

            case DT_TYPE_L_:
            case DT_TYPE_LU_:
                lval =
ldlong((void*) ((char*) xisam_stream->buffer_data + offset));
                    memcpy((void*) ((char*) xisam_stream->buffer_data
+ offset), &lval, LONG_SIZE);
                    break;

            case DT_TYPE_W_:
            case DT_TYPE_WU_:
                ival =
ldint((void*) ((char*) xisam_stream->buffer_data + offset));
                    memcpy((void*) ((char*) xisam_stream->buffer_data
+ offset), &ival, SHORT_SIZE);
                    break;

            default:
                break;
        }

        offset += column_da->length; /* move the offset */

    }

    return (GDB_OK_);
}

/*=====
 * Converts the outgoing buffer
=====*/
```

```
static GDB_STATUS store_buffer(XISAM_STREAM *xisam_stream)
{
    TABLE_DA *table_da = xisam_stream->table_da;
    COLUMN_DA *column_da;
    long offset = 0;
    long lval, col;
    short ival;

    for (col = 0; col < table_da->n_columns; col++) {

        column_da = &table_da->columns[col];

        /* check type */
        switch (column_da->datatype) {

            case DT_TYPE_L_:
            case DT_TYPE_LU_:
                memcpy(&lval,
(void*)((char*)xisam_stream->buffer_data + offset), LONG_SIZE);
                    stlong(lval,
(void*)((char*)xisam_stream->buffer_data + offset));
                    break;

            case DT_TYPE_W_:
            case DT_TYPE_WU_:
                memcpy(&ival,
(void*)((char*)xisam_stream->buffer_data + offset), SHORT_SIZE);
                    stint(ival,
(void*)((char*)xisam_stream->buffer_data + offset));
                    break;

            default:
                break;
        }

        offset += column_da->length; /* move the offset */
    }

    return (GDB_OK_);
}

/*=====
 * Creates a dynamic array of key structures based on table_da index
=====*/
```

```
static GDB_STATUS create_key_array(XISAM_STREAM *xisam_stream,
    TABLE_DA *table_da)
{
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    int i;
    struct keydesc *key;

    /* create dynamic array */
    ARR_NEW(table_da->n_indexs, &xisam_stream->index_array, 5,
    xisam_stream->memory);

    /* set each index */
    for (i = 0; i < table_da->n_indexs; i++) {
        key = MEMP_MALLOC(sizeof(struct keydesc),
    xisam_stream->memory);
        /* fill the key structure */
        if (set_key(xisam_connect, table_da, &table_da->indexes[i],
key) != GDB_OK_) {
            return (GDB_NOT_OK_);
        }

        /* add to array */
        xisam_stream->index_array[i] = key;
    }

    return (GDB_OK_);
}

/*=====
 * Set the key structure information based on index_da
 *=====
*/
static GDB_STATUS set_key(XISAM_CONNECT *xisam_connect,
    TABLE_DA *table_da,
    INDEX_DA *index_da,
    struct keydesc *key)
{
    int i;
    short offset = 0;
    COLUMN_DA *column_da;
    SEG_DA *seg_da;

    /* check uniqueness */
    if (GDB_INDEX_IS_UNIQUE(index_da) != 0) {
        key->k_flags = ISNODUPS;
    }
    else {
```

```
        key->k_flags = ISDUPS;
    }

/* check max segment */
if (index_da->n_segments > NPARTS) {
    set_error(xisam_connect, 0, "Error to many segments in
index", "SETKEY");
    return (GDB_NOT_OK_);
}

/* for each index segment */
key->k_nparts = (short)index_da->n_segments;
for (i = 0; i < key->k_nparts; i++) {

    /* get seg */
    seg_da = &index_da->segments[i];

    /* get column */
    column_da = &table_da->columns[seg_da->column_number];

    key->k_part[i].kp_start = offset;
    key->k_part[i].kp_leng = (short)column_da->length;

    /* determine type */
    switch (column_da->datatype) {
        case DT_TYPE_L_:
        case DT_TYPE_LU_:
            key->k_part[i].kp_type = LONGTYPE;
            break;

        case DT_TYPE_W_:
        case DT_TYPE_WU_:
            key->k_part[i].kp_type = INTTYPE;
            break;

        case DT_TYPE_F_:
            key->k_part[i].kp_type = FLOATTYPE;
            break;

        case DT_TYPE_D_:
            key->k_part[i].kp_type = DOUBLETYPE;
            break;

        case DT_TYPE_ISAM_DECIMAL_:
            key->k_part[i].kp_type = DECIMALTYPE;
            break;
    }
}
```

```
        default:
            key->k_part[i].kp_type = CHARTYPE;
            break;
    }

    /* order */
    if (GDB_SEG_IS_DESCENDING(seg_da)) {
        key->k_part[i].kp_type += ISDESC;
    }

} /* end for */

return (GDB_OK_);
}

/*=====
 * Creates a new table
 *=====
 */

static GDB_STATUS create_new_table(XISAM_STREAM *xisam_stream,
    TABLE_DA *table_da)
{
    XISAM_CONNECT *xisam_connect = xisam_stream->xisam_connect;
    long recsize;
    struct keydesc nokey;
    struct keydesc *key;
    long fd;
    long array_size;
    int index;
    int* error;
    GDB_STATUS status;

    nokey.k_nparts = 0; /* incase there is no primary key */
    array_size = ARR_SIZE(&xisam_stream->index_array);

    /* compute recsize */
    status = compute_recsize(&recsize, table_da);
    if (status != GDB_OK_) return status;

    /* determine primary key */
    if (array_size > 0) {
        key = xisam_stream->index_array[0]; /* primary key*/
    }
    else {
        key = &nokey; /* no primary key*/
    }
}
```

```
/* create the new table */
if ((fd = isbuild(table_da->path, recsize, key,
ISINOUT+ISEXCLLOCK+ISFIXLEN)) < 0) {
    error = is_errno(xisam_stream->fd);
    set_error(xisam_connect, *error, "Error creating table.",
"OPEN");
}

/* add the rest */
for (index = 1; index < array_size; index++) {
    key = xisam_stream->index_array[index];
    if (isaddindex(fd, key) < 0) {
        error = is_errno(xisam_stream->fd);
        set_error(xisam_connect, *error, "Error adding
index.", "OPEN");
    }
}

/* close */
if (isclose(fd) < 0) {
    error = is_errno(xisam_stream->fd);
    set_error(xisam_connect, *error, "Error closing table.",
"OPEN");
}

return (GDB_OK_);
}

=====
* Computes the record size based on the total column size.
*
* Note only creating fix length records
=====*/
```

```
static GDB_STATUS compute_recsize(long *recsize, TABLE_DA *table_da)

COLUMN_DA *column_da;
int column;
long size = 0;

for (column = 0; column < table_da->n_columns; column++) {

    column_da = &table_da->columns[column];

    switch (column_da->datatype) {
        case DT_TYPE_L_:
        case DT_TYPE_LU_:
```

```
        size += sizeof(LONGTYPE);
        break;

    case DT_TYPE_W_:
    case DT_TYPE_WU_:
        size += sizeof(INTTYPE);
        break;

    case DT_TYPE_F_:
        size += sizeof(FLOATTYPE);
        break;

    case DT_TYPE_D_:
        size += sizeof(DOUBLETYPE);
        break;

    default:
        /* CHARTYPE or DECIMAL */
        size += column_da->length;
        break;
    }
}

*recsize = size;

return (GDB_OK_);
}
```



## **Part B**

# **Appendix – Application Adapter Examples**



# Appendix B The Calc Sample Application Adapter

The Calc sample application adapter demonstrates how you design an application adapter that performs elementary arithmetic operations.

## The Application Adapter Definition

### The ADDON.DEF File

The ADDON.DEF definition for the Calc application adapter is as follows:

```
[calc]
type=adapter
INIT-FUNCTION=register_calc_adapter
SHAREABLE-NAME=C:\Program Files\Attunity\Connect\Samples\adapters\calc\calc.dll
```

### Binding Definition

The binding information for the calc sample application adapter is as follows:

```
<bindings>
  <binding name="name">
    <adapters name="binding_name">
      <adapter name="calc" connect="" type="calc"/>
    </adapters>
  </binding>
</bindings>
```

## ACX Schema for the Calc Adapter

The following is the ACX XML schema for the Calc sample adapter.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<adapter name="calc" description="Attunity Connect Administration
transactionLevelSupport="OPC" authenticationMechanism="basic-password"
maxActiveConnections="0" maxIdleTimeout="600" maxRequestSize="32000">

<!-- ++++++ -->
```

```
<!-- List all Interactions -->
<!-- ++++++-->

<!-- Simple Arithmetics only -->
<!-- ++++++-->
<!-- Some Binary Arithmetics -->
<!-- ..... -->

<interaction name="add" description="Add 2 numbers"
mode="sync-send-receive" input="binput" output="output" />

<interaction name="sub" description="Subtract 2 numbers"
mode="sync-send-receive" input="binput" output="output" />

<interaction name="mul" description="Multiply 2 numbers"
mode="sync-send-receive" input="binput" output="output" />

<interaction name="div" description="Divide a number by another"
mode="sync-send-receive" input="binput" output="output" />

<!-- Display -->
<!-- ++++++-->

<interaction name="display" description="Display a message in the output
stream" mode="sync-send-receive" input="inpmsg" output="outmsg" />
<!-- ..... -->
<!-- The Application Schema -->
<!-- ..... -->

<schema name="calc" version="1.0" header="calcdefs.h">
  <record name="binput">
    <field name="p1" type="int" />
    <field name="p2" type="int" />
  </record>

  <!-- Used for Binary arithmetics input -->
  <!-- ..... -->

  <record name="output">
    <field name="result" type="int" />
  </record>

  <!-- General purpose numeric output -->
  <!-- ..... -->

  <record name="inpmsg">
    <field name="m" type="string" />
  </record>
```

```

<record name="outmsg">
    <field name="m" type="string" nativeType="string" length="512" />
</record>
</schema>
</adapter>

```

## The C Code for the Calc Adapter

The following is the C source code for the Calc sample adapter:

```

***** This file contains a simple adapter that implements the
***** 4 elementary arithmetic operations. It provides the skeleton for
***** constructing an adapter.
***** Introduce adapter specific definitions
***** Implement all interactions|
***** */

/* Display a message */
/* ++++++ */
GAP_STATUS onDisplay(GAP_REQUEST *pGapRequest,
                     CALC_OUTMSG *pO,
                     CALC_INPMMSG *pI)
{
    sprintf(pO->sM, "\n%s\n", pI->pszM);
    return GAP_STATUS_OK;
}

```

```
/*
+-----+
|      Simple arithmetics
+-----+
*/
/* Add   */
/* +*** */
GAP_STATUS onAdd(GAP_REQUEST *pGapRequest, CALC_OUTPUT *pO,
CALC_BINPUT *pI)
{
    pO->iResult = pI->iP1 + pI->iP2;
    return GAP_STATUS_OK;
}

/* Subtract  */
/* +++++++ */
GAP_STATUS onSub(GAP_REQUEST *pGapRequest, CALC_OUTPUT *pO, CALC_BINPUT
*pI)
{
    pO->iResult = pI->iP1 - pI->iP2;
    return GAP_STATUS_OK;
}

/* Multiply  */
/* +++++++ */
GAP_STATUS onMul(GAP_REQUEST *pGapRequest, CALC_OUTPUT *pO,
CALC_BINPUT *pI)
{
    pO->iResult = pI->iP1 * pI->iP2;
    return GAP_STATUS_OK;
}

/* Divide   */
/* ++++++ */
GAP_STATUS onDiv(GAP_REQUEST *pGapRequest, CALC_OUTPUT *pO,
CALC_BINPUT *pI)
{
    if (pI->iP2 ==0) /* Exercise manual initiation of an exception */
    {
        GAP_SET_EXCEPTION(pGapRequest, "calc.onDiv",
"client.baddat",
"*** Attempting to divide by Zero **", NULL);
        return GAP_STATUS_ERROR;
    }
    else
    {
```

```
    pO->iResult = pI->iP1 / pI->iP2;
    return GAP_STATUS_OK;
}
}

/*
+-----+
-----+
Upon BIND - Register all functions implementing the interactions.
+-----+
-----+
*/
static GAP_STATUS onBind(GAP_REQUEST *pGapRequest,
void *pDummyOut, GAP_BIND *pBind)
{
    GAP_SET_INTERACTION_INFO(pGapRequest, "add", onAdd);
    GAP_SET_INTERACTION_INFO(pGapRequest, "sub", onSub);
    GAP_SET_INTERACTION_INFO(pGapRequest, "mul", onMul);
    GAP_SET_INTERACTION_INFO(pGapRequest, "div", onDiv);
    GAP_SET_INTERACTION_INFO(pGapRequest, "display", onDisplay);
    return GAP_STATUS_OK;
}

/*
+-----+
-----+
The adapter's INIT-FUNCTION
+-----+
*/
GAP_EXPORT register_calc_adapter(GAP_HANDLERS *pHandlers,
    GDB_HELP_FUNCTIONS *pHelp)
{
    GAP_INITIALIZE(pHelp);
    pHandlers->onBind = onBind;
/*
    pHandlers->onSetConnection = onSetConnection;
    pHandlers->onSetAutoCommit = onSetAutoCommit;
    pHandlers->onDisconnect = onDisconnect;
    pHandlers->onReauthenticate = onReauthenticate;
    pHandlers->onCleanConnection = onCleanConnection;
    pHandlers->onHoldConnection = onHoldConnection;
    pHandlers->onTransactionStart = onTransactionStart;
    pHandlers->onTransactionEnd = onTransactionEnd;
    pHandlers->onTransactionCommit = onTransactionCommit;
    pHandlers->onTransactionPrepare = onTransactionPrepare;
    pHandlers->onTransactionRollback = onTransactionRollback;
    pHandlers->onTransactionForget = onTransactionForget;
    pHandlers->onTransactionRecover = onTransactionRecover;
*/
}
```

```
/* No need for onExecute - onBind sets up automatic call of the interaction
function. */
/*
    pHandlers->onGetMetadataItem = onGetMetadataItem;
    pHandlers->onGetMetadataList = onGetMetadataList;
*/
    pHandlers->pAdapter = nv_init_calc_adapter();
        /* Eliminate the need for XML schema at runtime */
}
```

## Example Input to the Calc Demo Adapter

The following is a sample XML input file for the Calc adapter. This file specifies several basic arithmetic operations that are executed when the file is run with the Calc adapter.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<acx>
    <connect adapter="calc" />
    <execute interactionName="display">
        <inpmsg>
            <m>*** Welcome to CALC **</m>
        </inpmsg>
    </execute>
    <execute interactionName="display">
        <inpmsg>
            <m>**ADD interaction **</m>
        </inpmsg>
    </execute>
    <execute interactionName="add">
        <binput p1="1" p2="2" />
    </execute>
    <execute interactionName="display">
        <inpmsg>
            <m>**Sub interaction**</m>
        </inpmsg>
    </execute>
    <execute interactionName="sub">
        <binput p1="7" p2="2" />
    </execute>
    <execute interactionName="display">
        <inpmsg>
            <m>***Mul interaction***</m>
        </inpmsg>
    </execute>
    <execute interactionName="mul">
        <binput p1="5" p2="7" />
    </execute>
```

```
</execute>
<execute interactionName="display">
  <inpmsg>
    <m>***Div interaction***</m>
  </inpmsg>
</execute>
<execute interaction name="div">
  <binput p1="17" p2="6" />
</execute>
</acx>
```

## Example Output from the Calc Demo Adapter

The following file is output by the Calc adapter when it executes the sample XML input file defined above. You generate this output file by running the Calc adapter with the following command:

```
nav_util xml demo_calc.xml output_file
```

where:

**demo\_calc.xml** – is the XML input file shown in the previous section.

**output\_file** – is the file to which the output is written.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<acx type='response'>
  <connectResponse></connectResponse>
  <executeResponse>
    <outmsg m=' *** Welcome to CALC ** '>
    </outmsg>
  </executeResponse>
  <executeResponse>
    <outmsg m='** Add interaction **'>
    </outmsg>
  </executeResponse>
  <executeResponse>
    <output result='3'>
    </output>
  </executeResponse>
  <executeResponse>
    <outmsg m='** Sub interaction **'>
    </outmsg>
  </executeResponse>
  <executeResponse>
    <output result='5'>
    </output>
```

```
</executeResponse>
<executeResponse>
  <outmsg m='*** Mul interaction ***'>
    </outmsg>
</executeResponse>
<executeResponse>
  <output result='35'>
    </output>
</executeResponse>
<executeResponse>
  <outmsg m='*** Div interaction ***'>
    </outmsg>
</executeResponse>
<executeResponse>
  <output result='2'>
    </output>
</executeResponse>
</acx>
```

# Appendix C The Mailer Sample Adapter

The Mailer sample application adapter demonstrates how you design an application adapter that sends an email message using a compiled COM interface.

## The ADDON.DEF File

The ADDON.DEF definition for the Mailer application adapter is as follows:

```
[mailer]
type=adapter
INIT-FUNCTION=register_mailer_adapter
SHAREABLE-NAME=C:\Program Files\Attunity\Connect\Samples
\adapters\mailer\mailer.dll
```

## Binding Definition

The binding information for the Mailer sample application adapter is as follows:

```
<bindings>
  <binding name="name">
    <adapters name="binding_name">
      <adapter name="mailer" connect="" type="mailer"/>
    </adapters>
  </binding>
</bindings>
```

## The C++ Code for the Mailer Adapter

The following is the C++ source code for the Mailer sample application adapter:

```
*****
* M A I L E R . C P P
*
* This mailer adapter implements an interface to a
* COM ActiveX DLL that sends text as a mail message.
```

```
* The COM - ACsmtpDLL - is written in Visual Basic and
* carries out the SMTP protocol directly through a WINSOCK
* object (which is a COM by itself...)
* The mailer adapter demonstrates how to interact with
* a COM object of choice.
*
* THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
* KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR
* PURPOSE.
*
* Copyright (c) 2000 Attunity Ltd. All Rights Reserved.
* ****
#include <stdio.h>
#include "gap.h"

GDB_HELP_DEFINE;

/* The mailer prototype */
/* ++++++ */

int SMTPMailer(char *SendServ,char *SendTo, char *SendFrom,
               char *SendSbj, char *SendTxt,char *RetTxt);
/*
-----+
|   Introduce adapter specific definitions |
-----+
*/
#include "mailerinit.h"

/*
-----+
|   I m p l e m e n t   a l l   i n t e r a c t i o n s   |
-----+
*/

*/
/* Display a message */
/* ++++++ */
GAP_STATUS onDisplay(GAP_REQUEST *pGapRequest,
                     MAILER_OUTMSG *pO,
                     MAILER_INPMMSG *pI)
{
    sprintf(pO->sM, "\n%s\n", pI->pszM);
    return GAP_STATUS_OK;
}
/*
```

```
+-----+
| Invoke The Mailer
+-----+
*/
GAP_STATUS onSmtpMailer(GAP_REQUEST *pGapRequest,
                        MAILER_OUTMSG *pO,
                        MAILER_SMTPDTLS *pI)
{
    char RetTxt[196];
    int MailerRetCode;

    MailerRetCode = SMTPMailer(pI->pszSendserv,
                               pI->pszSendto,
                               pI->pszSendfrom,
                               pI->pszSendsbj,
                               pI->pszSendtxt,
                               RetTxt);
    if (MailerRetCode<0)
    {
        GAP_SET_EXCEPTION(pGapRequest, "mailer.onSmtpMailer",
                           "client.failure", RetTxt, NULL);
        return GAP_STATUS_ERROR;
    }
    else
    {
        sprintf(pO->sM, "%s", "OK");
        return GAP_STATUS_OK;
    }
}

/*
+-----+
| Upon BIND - Register all functions
| implementing the interactions.
+-----+
*/
static GAP_STATUS onBind(GAP_REQUEST *pGapRequest,
                        void *pDummyOut, GAP_BIND *pBind)
{
    GAP_SET_INTERACTION_INFO(pGapRequest, "smtpmailer", onSmtpMailer);
    GAP_SET_INTERACTION_INFO(pGapRequest, "display", onDisplay);
    return GAP_STATUS_OK;
}
/*
+-----+
|
```

The adapter's INIT-FUNCTION

Note: extern C is surrounding the function, thus suppressing the C++ decoration that may do bad things while exporting symbols to the world.

```
+-----+  
*/  
extern "C" {  
    GAP_EXPORT register_mailer_adapter(GAP_HANDLERS *,  
                                         GDB_HELP_FUNCTIONS *);  
}  
GAP_EXPORT register_mailer_adapter(GAP_HANDLERS *pHandlers,  
                                         GDB_HELP_FUNCTIONS *pHelp)  
{  
    GAP_INITIALIZE(pHelp);  
    pHandlers->onBind = onBind;  
    pHandlers->pAdapter = nv_init_mailer_adapter();  
    /* Eliminate the need for XML schema at runtime */  
}  
/*  
+-----+
```

SMTPMailer.cpp

Func: Invoke a COM ActiveX DLL (originated in Visual Basic)  
 - ACsmtpDLL - to send text as a mail message.

Calling sequence:

```
{int_var} = SMTPMailer(SendServ,SendTo, SendFrom,  
                         SendSbj,SendTxt, RetTxt);  
+-----+
```

```
/*  
#import "d:\winnt\system32\ACsmtpDLL.dll" no_namespace
```

```
#define ACsmtpProgID L"ACsmtpDLL.SMTP"  
/* This is the related registered Information */  
#define SMTPSTAMP "%SMTPMailer: "  
#define SMTPOk0  
#define SMTPNok (-1)  
  
int SMTPMailer(char *SendServ,char *SendTo, char *SendFrom,  
               char *SendSbj, char *SendTxt,char *RetTxt)  
{  
/*  
+-----+
```

```
|      S e t      s o m e      l o c a l      v a r i a b l e s      |
+-----+
*/
CLSID ACsmtpClSID;

HRESULT hr; // A handy variable for COM call results
int smtp_stat;
int ret_stat;
char RetMsg[196];
_variant_t vServ(SendServ);
_variant_t vSendTo(SendTo);
_variant_t vSendFrom(SendFrom);
_variant_t vSendTxt(SendTxt);
_variant_t vSendSbj(SendSbj);
_variant_t vRetMsg(RetMsg);
/*
+-----+
|      I n t e r a c t      W i t h      S M T P      m a i l e r      |
+-----+
*/
// 1st step - Query for the Class IDs.
// ++++++
hr = CLSIDFromProgID (ACsmtpProgID,&ACsmtpClSID);
if (hr!=S_OK)
{
    char prgid[64];
    wcstombs( prgid, ACsmtpProgID, sizeof prgid );
    sprintf(RetTxt,"%s CLSIDFromProgID failed - %s",SMTPSTAMP,prgid);
    return(SMTPNok);
}
// 2nd step - Initialize COM
// ++++++
hr = CoInitialize(NULL);
if (FAILED(hr))
{
    sprintf(RetTxt,"%s CoInitialize failed!!",SMTPSTAMP);
    return(SMTPNok);
}
_SMTPPtr SMTP(ACsmtpClSID);
// 3rd step - Accept User Mailing info
// ++++++
SMTP->PutMailServer(&vServ.bstrVal);
SMTP->PutMailTo(&vSendTo.bstrVal);
SMTP->PutMailFrom(&vSendFrom.bstrVal);
```

```
SMTP->PutMailSubj (&vSendSbj.bstrVal) ;
SMTP->PutMailMessage (&vSendTxt.bstrVal) ;
// 4th step - Do the job
// ++++++
smtp_stat = SMTP->SendMail() ;
// 5th step - Terminate
// ++++++
if (smtp_stat<0)
{
    char TmpMsg[256] ;
    vRetMsg.bstrVal = SMTP->GetTermMessage() ;
    wcstombs(TmpMsg,vRetMsg.bstrVal,sizeof TmpMsg) ;
    ret_stat= SMTPNok ;
    sprintf(RetTxt,"%s In SendMail() - %s. Status is - %d"
           ,SMTPSTAMP,TmpMsg,smtp_stat) ;
}
else
{
    ret_stat= SMTPOk ;
    RetTxt[0]='\0' ;
}
SMTP = NULL;
CoUninitialize();
return(ret_stat);
}
```

## ACX Schema for Mailer Adapter

The following is the ACX XML schema for the Mailer sample adapter:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<adapter name="mailer" description="Attunity Connect Mailer adapter"
  transactionLevelSupport="OPC" authenticationMechanism="basic-password"
  maxActiveConnections="0" maxIdleTimeout="600" maxRequestSize="32000">

<!-- ++++++ -->
<!-- L i s t a l l I n t e r a c t i o n s -->
<!-- ++++++ -->
<!-- S M T P M a i l e r -->
<!-- ++++++ -->

<interaction name="smtpmailer" description="Send a mail using Native
  SMTP" mode="sync-send-receive" input="smtpdtls" output="outmsg" />

<!-- D i s p l a y -->
<!-- ++++++ -->

<interaction name="display" description="Display a message in the
```

```
output stream" mode="sync-send-receive" input="inpmsg" output="outmsg" />

<!-- ..... -->
<-- The Application Schema -->
<-- ..... -->

<schema name="mailer" version="1.0" header="mailerdefs.h">
  <record name="inpmsg">
    <field name="m" type="string" />
  </record>
  <record name="outmsg">
    <field name="m" type="string" nativeType="string" length="512" />
  </record>

  <!-- Demonstrate the SMTP mailer -->
  <!-- ++++++ -->

  <record name="smtpdtsl">
    <field name="sendserv" type="string" required="true" />
    <field name="sendto" type="string" required="true" />
    <field name="sendfrom" type="string" required="true" />
    <field name="sendsbj" type="string" required="true" />
    <field name="sendtxt" type="string" required="true" />
  </record>
</schema>
</adapter>
```

## Example Input to the Mailer Demo Adapter

The following is a sample XML input file for the Mailer adapter. This file specifies the message sent by the Mailer adapter.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<acx>
<connect adapter="mailer" />
<execute interactionName="display">
  <inpmsg>
    <m>*** Welcome to the MAILER adapter *** Message mailing using a
    private ACsmtpDLL COM Note: ACsmtpDLL.dll must reside at data driver
      Windows system directory and registered properly
      (regsvr32 ACsmtpDLL.dll)
    </m>
  </inpmsg>
</execute>
<execute interactionName="smtpmailer">
  <smtpdtsl>
```

```
<sendserv>mailhost.acme.com</sendserv>
<sendto>scott@acme.com</sendto>
<sendfrom>jane@acme.com</sendfrom>
<sendsbj>A message from Mailer.</sendsbj>
<sendtxt>Hello there Dec-2000 This mail message has been sent through
Attunity Adapter interface to an SMTP/COM ACTIVEEx DLL originated
in Visual Basic. Regards Attunity team.</sendtxt>
</smtpdts>
</execute>
</acx>
```

## Example Output from the Mailer Demo Adapter

The following output is generated by the Mailer adapter when it executes the sample XML input file defined above. You generate this output file by running the Mailer adapter with the following command:

```
nav_util xml demo_mailer.xml output_file
```

where:

**demo\_mailer.xml** – is the XML input file shown in the previous section.

**output\_file** – is the file to which the output is written.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<acx type='response'>
  <connectResponse>
  </connectResponse>
  <executeResponse>
    <outmsg m='*** Welcome to the MAILER adapter ***
Message mailing using a private ACsmtpDLL COM
Note: ACsmtpDLL.dll must reside at the Windows
system directory and registered properly
(regsvr32 ACsmtpDLL.dll)'>
    </outmsg>
  </executeResponse>
  <executeResponse>
    <outmsg m='OK'>
    </outmsg>
  </executeResponse>
</acx>
```

## **Part C**

# **Appendix – User Defined Data Type Example**



## Appendix D Sample RVMS Data Type

This sample defines a custom data type that is identical to VMS\_DATE except the bytes are in reverse order.

The new data type is based on Attunity Connect's standard data type for VMS\_DATE and is named RVMS\_DATE, which is the value you supply when defining a column in ADD metadata syntax. The data type definition provides the required conversion functions to and from a string, as well as optional conversion functions to and from a VMS\_DATE.

- ❖ The to and from string conversions in this example are implemented by using the UDT\_CONVERT function (from the help function vector), taking advantage of the fact that Attunity Connect knows how to convert a VMS\_DATE to and from a string.

To integrate this data type definition with the Attunity Connect environment, you must register add\_rvms\_date\_dt to Attunity Connect. (For details, see page 250.)

### The RVMS Data Type Source Code

```
#include "dbgdb.h"
#include "dtypeid.h"

GDB_HELP_DEFINE;

#define NULL 0

/*****
* flip_vms_date: conversion routine to switch bytes
* of VMS date
*****/

static void flip_vms_date(
    GDB_VAL_DESC *vd_desc_t,
    GDB_VAL_DESC *vd_desc_s)
{
    long i ;
    char *src = vd_desc_s->data ;
    char *trg = vd_desc_t->data ;
```

```
        for (i=0; i<8; i++)
            trg[i] = src[7-i] ;
    }

/*****
* udt_rvd_to_string: conversion routine for RVMS
* data type to string
*****/

static UDT_STATUS udt_rvd_to_string(
    GDB_VAL_DESC *str_desc,
    GDB_VAL_DESC *rvd_desc)
{
    char temp_buffer[8] ;
    GDB_VAL_DESC vd_desc ;

    vd_desc.id = DT_TYPE_ADT_ ;
    vd_desc.size = 8 ;
    vd_desc.width = 0 ;
    vd_desc.scale = 0 ;
    vd_desc.data = temp_buffer ;

    flip_vms_date(&vd_desc, rvd_desc) ;

    UDT_CONVERT(str_desc,&vd_desc) ;
    return (UDT_OK_) ;
}

/*****
* udt_rvd_from_string: conversion routine from
* string to RVMS data type
*****/

static UDT_STATUS udt_rvd_from_string(
    GDB_VAL_DESC *rvd_desc,
    GDB_VAL_DESC *str_desc)
{
    char temp_buffer[8] ;
    GDB_VAL_DESC vd_desc ;

    vd_desc.id = DT_TYPE_ADT_ ;
    vd_desc.size = 8 ;
    vd_desc.width = 0 ;
    vd_desc.scale = 0 ;
    vd_desc.data = temp_buffer ;

    UDT_CONVERT(&vd_desc, str_desc) ;
```

```
        flip_vms_date(rvd_desc, &vd_desc) ;
        return (UDT_OK_) ;
    }

/*****
* udt_rvd_to_base: conversion routine from RVMS
* data type to base type
*****/

static UDT_STATUS udt_rvd_to_base(
    GDB_VAL_DESC *vd_desc,
    GDB_VAL_DESC *rvd_desc)
{
    flip_vms_date(vd_desc, rvd_desc) ;
    return (UDT_OK_) ;
}

/*****
* udt_rvd_from_base: conversion routine from base
* type to RVMS data type
*****/

static UDT_STATUS udt_rvd_from_base(
    GDB_VAL_DESC *rvd_desc,
    GDB_VAL_DESC *vd_desc)
{
    flip_vms_date(rvd_desc, vd_desc) ;
    return (UDT_OK_) ;
}

/*****
* add_rvms_date_dt: sets up the function pointer
* for this UDT data type
*****/

void add_rvms_date_dt(GDB_HELP_FUNCTIONS *help_fncts)
{
    static UDT_DATATYPE udt_datatype ;
    gdb_help = help_fncts;

    udt_datatype.id = 201 ;
    udt_datatype.base_id = DT_TYPE_ADT_ ;
    udt_datatype.name = "RVMS_DATE" ;
    udt_datatype.size = 8 ;
    udt_datatype.to_string = udt_rvd_to_string ;
    udt_datatype.from_string = udt_rvd_from_string ;
}
```

```
    udt_datatype.to_base = udt_rvd_to_base ;
    udt_datatype.from_base = udt_rvd_from_base ;
    udt_datatype.compare = NULL ;
    udt_datatype.get_last_error = NULL ;
    udt_datatype.attributes = 0 ;

    UDT_SET_DATE(&udt_datatype) ;
    UDT_ADD_DATATYPE(&udt_datatype) ;
}
```

## The Data Type ADDON.DEF Entry

To specify this sample data type in the Attunity Connect environment, create a custom define file **dtxxx.def** as follows:

```
[RVMS_DATE]
TYPE=DATATYPE or STARTUP
INIT-FUNCTION=add_rvms_date_dt
SHAREABLE-NAME=NVDT_XXX
```

Merge this definition with any pre-existing definitions in the **addon.def** file, using the following command:

```
nav_util addon dtxxxx.def
```

Prior to accessing data with this data type using Attunity Connect, you must make sure that Attunity Connect can access the DLL that contains the data type code, identified as NVDT\_XXX in **addon.def**. See "System Notes" on page 253 for platform-specific information about this task.

## **Part D**

# **Appendix – System Notes and SDK Data Structures**



# Appendix E System Notes

The type of platform on which Attunity Connect runs affects the SHAREABLE-NAME entry you specify for a data driver (see page 25), startup function (see page 151), or ODBC driver (see page 83).

<b>Windows</b>	<p>The <i>image</i> value must be a full DLL name, preferably with the directory as once the directory is specified, all dependent DLLs are first looked up in that directory.</p> <p>a full DLL name, preferably with the directory as once the directory is specified, all dependant DLLs are first looked up in that directory</p>
<b>UNIX</b>	<p>The <i>image</i> value must be one of the following:</p> <ul style="list-style-type: none"><li>■ A UNIX environment variable which translates to the full path of a shared library. The variable name is translated before accessing the data.</li><li>■ The full path of a shared library.</li><li>■ The name of a shared library in the directory \$NAVROOT\lib, without any path or extension (Attunity Connect adds the necessary suffix, to simplify different UNIX notations).</li></ul> <p>The library that you specify may be loaded with different options depending on the particular UNIX operating system. The following are some examples; consult the system's documentation for details:</p> <p><b>AIX:</b> Using the “load” system function with settings: Flags=0 LibraryPath=NULL</p> <p><b>HP:</b> Using the “shl_load” system function with settings: Flags=BIND_DEFERRED   IND_NONFATAL   BIND_FIRST   BIND_VERBOSE Address=0</p> <p><b>DGUX, DEC-UNIX, SUN:</b> Using the “dlopen”+“dlsym” system functions with setting: Flags=RTDL_NOW</p>
<b>OpenVMS</b>	<p>The <i>image</i> value must be a logical name which translates to the complete directory and file specification of a shareable image. The logical name is translated before accessing the data.</p>

**Tandem**

Tandem platforms do not provide support for DLLs. To incorporate custom code into the Attunity Connect environment on a Tandem platform, you must replace the supplied NAV\_UTIL program with a copy that includes the data drivers, data types, and startup code. To use the Developer SDK on a Tandem platform, do the following:

1. Create a custom version of the main() function for the NAV\_UTIL program.
2. In the custom main() function, register all custom functions that are referenced in addon.def. Use the following Attunity Connect function to register the functions:

```
long nav_register_function(STRING fnc_name,  
                           STRING fnc_class, FNC_PT fnc_pt)
```

where:

**fnc\_name** – The name of the “external” function to be registered (that is, the LOAD function for a driver, the registration function for a data type, or a startup function).

**fnc\_class** – The name of the group you want the function to be in (this is equivalent to the DLL module name on other platforms).

**fnc\_pt** – A pointer to the user function.

3. Statically link the custom functions with the Attunity Connect library LIBNAVA, located in the subvolume where Attunity Connect is installed, to create a custom version of the NAV\_UTIL program.
4. In **addondef**, specify the value of **fnc\_name** as the *function* entry (INIT-FUNCTION) and the value of **fnc\_class** as the *image* entry (SHAREABLE-NAME).

**Example**

```
#include "utlmain.h"  
main(int argc, char *argv[])  
{  
    nav_register_function("db_file_get_functions",  
                          "NVDB_FILE", db_file_get_functions) ;  
    nav_register_function("add_rvms_date_dt",  
                          "MY_DATATYPES", add_rvms_date_dt) ;  
  
    return(nav_util_main(argc, argv));  
}
```

# Appendix F Data Structures

This section describes data structures defined in the Developer SDK:

- Function Structures
- Data Buffer Structures
- Metadata Structures
- Filter Structure
- Data Type Structure

This section also describes the following:

- Attributes – To provide support for various properties that you may need in a data driver or data type.
- Return Status Codes
- ❖ All of the data structures and macros described here are defined in the header files dbgdb.h and odbcfnsh.h (see page 19).

All structures need to be initialized to zero, using memset, prior to loading them.

## Function Structures

To identify the functions used for data drivers and data types, populate the following structures:

- GDB\_DB\_FUNCTIONS
- UDT\_DATATYPE (See "Data Type Structure" on page 290)

Attunity Connect populates the following structure to identify the support functions available for data drivers and data types:

- GDB\_HELP\_FUNCTIONS

To identify the functions used for a custom ODBC driver, you can optionally populate the following structure:

- ODBC\_FUNCTIONS

### GDB\_DB\_FUNCTIONS

This structure lists all of the driver API functions and attributes.

The driver can omit assignments in the structure for any functions that are not used. For example, if the driver is read-only, you do not need to assign a value to the `io_update_row` structure member.

Attunity Connect ignores any reference to a function that is missing from the structure. For example, the `UNLOAD` function is optional; Attunity Connect calls it only if it has been specified in the structure for the driver.

```
struct GDB_DB_FUNCTIONS {
    long version;
    long attributes;
    long bookmark_size;
    char sql_syntax_name[80+1];
    void *attrib_unused[10];
    GDB_FNC_GET_LAST_ERRORio_get_last_error;
    GDB_FNC_UNLOADio_unload;
    GDB_FNC_CONNECTio_connect;
    GDB_FNC_DISCONNECTio_disconnect;
    GDB_FNC_SET_CONNECTio_set_connect;
    void *connect_unused[5];
    GDB_FNC_GET_ITEM_LISTio_get_item_list;
    GDB_FNC_DESCRIBE_TABLEio_describe_table;
    GDB_FNC_DESCRIBE_SPiо_describe_sp;
    GDB_FNC_GET_EXTENDED_METADATAio_get_extended_metadata;
    GDB_FNC_GET_FILE_INFOio_get_file_info;
    void *metadata_unused[4];
    GDB_FNC_OPENio_open;
    GDB_FNC_SET_BUFFERio_set_buffer;
    GDB_FNC_SET_INDEXio_set_index;
    GDB_FNC_SET_PARAMSio_set_params;
    GDB_FNC_SET_FILTERio_set_filter;
    GDB_FNC_SET_FILTER_PARAMSio_set_filter_params;
    GDB_FNC_SET_BMio_set_bm;
    GDB_FNC_FETCHio_fetch;
    GDB_FNC_CLOSEio_close;
    GDB_FNC_REWINDio_rewind;
    GDB_FNC_CONVERT_BM_DATAio_convert_bm_data;
    void *stream_unused[5];
    GDB_FNC_ADD_ROWio_add_row;
    GDB_FNC_UPDATE_ROWio_update_row;
    GDB_FNC_DELETE_ROWio_delete_row;
    GDB_FNC_LOCK_ROWio_lock_row;
    GDB_FNC_UNLOCK_ROWio_unlock_row;
    void *update_unused[5];
    GDB_FNC_START_TRANSACTIONio_start_transaction;
    GDB_FNC_COMMIT_TRANSACTIONio_commit_transaction;
    GDB_FNC_ROLLBACK_TRANSACTIONio_rollback_transaction;
```

```

GDB_FNC_START_DISTRIBUTED_TRANSACTION
io_start_distributed_transaction;
GDB_FNC_PREPARE_COMMIT_TRANSACTION io_prepare_commit_transaction;
GDB_FNC_RECOVER_TRANSACTION io_recover_transaction;
void *transaction_unused[5];
GDB_FNC_CREATE_TABLE io_create_table;
GDB_FNC_CREATE_INDEX io_create_index;
GDB_FNC_DROP_TABLE io_drop_table;
GDB_FNC_DROP_INDEX io_drop_index;
void *ddl_unused[5];
GDB_FNC_BLOB_OPEN io_blob_open;
GDB_FNC_BLOB_READ io_blob_read;
GDB_FNC_BLOB_WRITE io_blob_write;
GDB_FNC_BLOB_SEEK io_blob_seek;
GDB_FNC_BLOB_CLOSE io_blob_close;
void *blob_unused[5];
GDB_FNC_PREPARE_COMMAND io_prepare_command;
GDB_FNC_DESCRIBE_COMMAND io_describe_command;
GDB_FNC_EXECUTE_IMMEDIATE io_execute_immediate;
GDB_FNC_EXECUTE_WITH_PARAMS io_execute_with_params;
GDB_FNC_FREE_COMMAND io_free_command;
GDB_FNC_GET_DATE_LITERAL io_get_date_literal;
void *command_unused[5];
GDB_FNC_DESCRIBE_CHAPTER io_describe_chapter;
GDB_FNC_OPEN_CHAPTER io_open_chapter;
GDB_FNC_SET_ACTIVE_CHAPTER io_set_active_chapter;
void *chapter_unused[5];
}

```

**version** The version number of the GDB API that is used with this driver. Attunity Connect checks this version number to ensure compatibility between the structures defined in the GDB API version when you built the driver and those defined in the current version.

Use the macro GDB\_INITIALIZE in the LOAD function to set this element equal to GDB\_VERSION\_NUMBER\_, which is defined in the header file for the GDB API.

**attributes** The GDB API supports various attributes related to the general functioning of the driver. In the GDB\_DB\_FUNCTIONS structure, each individual attribute setting is mapped to a particular bit of the attributes element. The driver-level attribute's are listed below. (For details, see page 294.)

Attribute Name	Description
GDB_M_ADD_	Indicates whether the driver metadata is stored in Attunity Connect Data Dictionary (ADD).

Attribute Name	Description
GDB_M_SUPPORT_COLUMN_LIST_	Indicates whether the driver can perform read and update operations based on a requested list of columns, or only on entire records.
GDB_M_INSERT_ALL_COLUMNS_	For drivers that support column lists, this attribute indicates whether the driver requires a complete record buffer of all columns when inserting a new row, despite the driver's ability to work at the column level for other operations.
GDB_M_SUPPORT_TRANS_	Indicates whether the driver supports transactions.
GDB_M_SUPPORT_UPDATE_	Indicates whether the driver allows update operations.
GDB_M_SUPPORT_MULTI_THREAD_	Indicates whether the driver is multi-thread-safe (the driver can properly handle separate threads for multiple clients accessing a shared server process at the same time).
GDB_M_SUPPORT_PASSTHRU_COMMAND_	Indicates whether the driver supports PASSTHRU command syntax.
GDB_M_CASE_SENSITIVE_	Indicates whether the driver is case sensitive to application data and metadata names (such as column names and table names).
GDB_M_DISABLE_GST_	Indicates whether the driver supports parallel execution by the client application and the server query processor.
GDB_M_SUPPORT_FILTERS_	Indicates whether the driver supports filtering expressions that do not necessarily apply to an index being used to access data.
GDB_M_SUPPORT_FILTER_PARAMS_	Indicates whether the driver supports parameters passed in filter expressions.
GDB_M_FILTER_WO_INDEX_	Indicates whether the driver uses information only from the filter expression and not an index, to filter data. When set, the Attunity Connect query processor performs a full table read with a filter passed to the driver on all references to a table.
GDB_M_FILTER_WO_SEGMENTS_	Indicates whether the driver requires separate filtering of data that would return segments of an index.
GDB_M_SUPPORT_EXTENDED_NAMES_	Indicates whether the driver uses Attunity Connect extended metadata to support virtual tables and arrays, regardless of whether the driver uses ADD for other metadata.

<b>Attribute Name</b>	<b>Description</b>
GDB_M_SUPPORT_SQL_	Indicates whether the driver supports SQL command syntax.
GDB_M_SUPPORT_ABS_OFFSET_	Indicates whether driver supports use of absolute offset, defined as an element in the COLUMN_DA structure.
<b>bookmark_size</b>	<p>Bookmarks uniquely identify rows. You must supply bookmark_size in the following cases:</p> <ul style="list-style-type: none"> <li>■ The driver allows read-write operations.</li> <li>■ The driver supports arrays.</li> </ul>
	<p>In cases where all tables accessed by the driver have the same bookmark size, you can use the macro GDB_SET_BOOKMARK_SIZE at the driver level to set the bookmark_size for all tables. Or, you can set a different bookmark_size when you specify the TABLE_DA structure for a given table.</p>
<b>sql_syntax_name</b>	<p>This element applies to relational drivers, which include commands as part of the table definition. This element defines the type of syntax the Query Processor uses when generating SQL commands. The name that you specify must be included in the Attunity Connect syntax file.</p>
<b>io_get_last_error</b>	<p>The name of the driver function that corresponds to the GET_LAST_ERROR routine (page 76) in the GDB API.</p>
<b>io_unload</b>	<p>The name of the driver function that corresponds to the UNLOAD routine (page 35) in the GDB API.</p>
<b>io_connect</b>	<p>The name of the driver function that corresponds to the CONNECT routine (page 35) in the GDB API.</p>
	<p>This element is required.</p>
<b>io_disconnect</b>	<p>The name of the driver function that corresponds to the DISCONNECT routine (page 38) in the GDB API.</p>
<b>io_set_connect</b>	<p>The name of the driver function that corresponds to the SET_CONNECT routine (page 36) in the GDB API.</p>
<b>io_get_item_list</b>	<p>The name of the driver function that corresponds to the GET_ITEM_LIST routine (page 39) in the GDB API.</p>
<b>io_describe_table</b>	<p>The name of the driver function that corresponds to the DESCRIBE_TABLE routine (page 40) in the GDB API.</p>

<b>io_describe_sp</b>	The name of the driver function that corresponds to the DESCRIBE_SP routine (page 41) in the GDB API.
<b>io_get_extended_metadata</b>	The name of the driver function that corresponds to the GET_EXTENDED_METADATA routine (page 42) in the GDB API.
<b>io_get_file_info</b>	The name of the driver function that corresponds to the GET_FILE_INFO routine (page 43) in the GDB API.
<b>io_open</b>	The name of the driver function that corresponds to the OPEN routine (page 44) in the GDB API.  This element is required.
<b>io_set_buffer</b>	The name of the driver function that corresponds to the SET_BUFFER routine (page 45) in the GDB API.  This element is required.
<b>io_set_index</b>	The name of the driver function that corresponds to the SET_INDEX routine (page 46) in the GDB API.
<b>io_set_params</b>	The name of the driver function that corresponds to the SET_PARAMS routine (page 51) in the GDB API.
<b>io_set_filter</b>	The name of the driver function that corresponds to the SET_FILTER routine (page 48) in the GDB API.
<b>io_set_filter_params</b>	The name of the driver function that corresponds to the SET_FILTER_PARAMS routine (page 50) in the GDB API.
<b>io_set_bm</b>	The name of the driver function that corresponds to the SET_BM routine (page 51) in the GDB API.
<b>io_fetch</b>	The name of the driver function that corresponds to the FETCH routine (page 53) in the GDB API.  This element is required.
<b>io_close</b>	The name of the driver function that corresponds to the CLOSE routine (page 54) in the GDB API.  This element is required.
<b>io_rewind</b>	The name of the driver function that corresponds to the REWIND routine (page 55) in the GDB API.

	This element is required.
<b>io_convert_bm_data</b>	The name of the driver function that corresponds to the CONVERT_BM_DATA routine (page 53) in the GDB API.
<b>io_add_row</b>	The name of the driver function that corresponds to the ADD_ROW routine (page 56) in the GDB API.
<b>io_update_row</b>	The name of the driver function that corresponds to the UPDATE_ROW routine (page 56) in the GDB API.
<b>io_delete_row</b>	The name of the driver function that corresponds to the DELETE_ROW routine (page 57) in the GDB API.
<b>io_lock_row</b>	The name of the driver function that corresponds to the LOCK_ROW routine (page 58) in the GDB API.
<b>io_unlock_row</b>	The name of the driver function that corresponds to the UNLOCK_ROW routine (page 58) in the GDB API.
<b>io_start_transaction</b>	The name of the driver function that corresponds to the START_TRANSACTION routine (page 59) in the GDB API.
<b>io_commit_transaction</b>	The name of the driver function that corresponds to the COMMIT_TRANSACTION routine (page 60) in the GDB API.
<b>io_rollback_transaction</b>	The name of the driver function that corresponds to the ROLLBACK_TRANSACTION routine (page 60) in the GDB API.
<b>io_start_distributed_transaction</b>	<ul style="list-style-type: none"><li>❖ <i>For future use:</i> This structure element will be implemented in an upcoming version of the Developer SDK.</li></ul> <p>The name of the driver function that corresponds to the START_DISTRIBUTED_TRANSACTION routine (page 61) in the GDB API.</p>
<b>io_prepare_commit_transaction</b>	<ul style="list-style-type: none"><li>❖ <i>For future use:</i> This structure element will be implemented in an upcoming version of the Developer SDK.</li></ul> <p>The name of the driver function that corresponds to the PREPARE_COMMIT_TRANSACTION routine (page 61) in the GDB API.</p>

<b>io_recover_transaction</b>	❖ <i>For future use:</i> This structure element will be implemented in an upcoming version of the Developer SDK.
	The name of the driver function that corresponds to the RECOVER_TRANSACTION routine (page 61) in the GDB API.
<b>io_create_table</b>	The name of the driver function that corresponds to the CREATE_TABLE routine (page 62) in the GDB API.
<b>io_create_index</b>	The name of the driver function that corresponds to the CREATE_INDEX routine (page 63) in the GDB API.
<b>io_drop_table</b>	The name of the driver function that corresponds to the DROP_TABLE routine (page 64) in the GDB API.
<b>io_drop_index</b>	❖ <i>For future use:</i> This structure element will be implemented in an upcoming version of the Developer SDK.
	The name of the driver function that corresponds to the DROP_INDEX routine (page 65) in the GDB API.
<b>io_blob_open</b>	The name of the driver function that corresponds to the BLOB_OPEN routine (page 65) in the GDB API.
<b>io_blob_read</b>	The name of the driver function that corresponds to the BLOB_READ routine (page 66) in the GDB API.
<b>io_blob_write</b>	The name of the driver function that corresponds to the BLOB_WRITE routine (page 67) in the GDB API.
<b>io_blob_seek</b>	❖ <i>For future use:</i> This structure element will be implemented in an upcoming version of the Developer SDK.
<b>io_blob_close</b>	The name of the driver function that corresponds to the BLOB_CLOSE routine (page 68) in the GDB API.
<b>io_prepare_command</b>	The name of the driver function that corresponds to the PREPARE_COMMAND routine (page 69) in the GDB API.
<b>io_describe_command</b>	The name of the driver function that corresponds to the DESCRIBE_COMMAND routine (page 69) in the GDB API.
<b>io_execute_immediate</b>	The name of the driver function that corresponds to the EXECUTE_IMMEDIATE routine (page 70) in the GDB API.

**io\_execute\_with\_params**

The name of the driver function that corresponds to the EXECUTE\_WITH\_PARAMS routine (page 71) in the GDB API.

**io\_free\_command**

The name of the driver function that corresponds to the FREE\_COMMAND routine (page 72) in the GDB API.

**io\_get\_date\_literal**

The name of the driver function that corresponds to the GET\_DATE\_LITERAL routine (page 72) in the GDB API.

**io\_describe\_chapter**

The name of the driver function that corresponds to the DESCRIBE CHAPTER routine (page 73) in the GDB API.

**io\_open\_chapter**

The name of the driver function that corresponds to the OPEN CHAPTER routine (page 74) in the GDB API.

**io\_set\_active\_chapter**

The name of the driver function that corresponds to the SET\_ACTIVE CHAPTER routine (page 75) in the GDB API.

## GDB\_HELP\_FUNCTIONS

Attunity Connect allocates and populates this structure, including the names of the auxiliary functions supplied with the Developer SDK.

```
typedef struct {
    long version;
    long gdb_debug_mode;
    GDB_FNC_MEMP_NEWnv_memp_new;
    GDB_FNC_MEMP_MALLOCnv_memp_malloc;
    GDB_FNC_MEMP_MALLOC8nv_memp_malloc8;
    GDB_FNC_MEMP_FREEEnv_memp_free;
    GDB_FNC_MEMP_FREE_POOLnv_memp_free_pool;
    GDB_FNC_MEMP_DELETEEnv_memp_delete;
    void *memp_unused[5];
    GDB_FNC_ARR_NEWnv_arr_new;
    GDB_FNC_ARR_RESETnv_arr_reset;
    GDB_FNC_ARR_SIZEnv_arr_size;
    GDB_FNC_ARR_DELETEEnv_arr_delete;
    GDB_FNC_ARR_SET_ENTRYnv_arr_set_entry;
    GDB_FNC_ARR_INSERTnv_arr_insert;
    GDB_FNC_ARR_ADDnv_arr_add;
    GDB_FNC_ARR_COPYnv_arr_copy;
    GDB_FNC_ARR_FREEEnv_arr_free;
    GDB_FNC_ARR_BINSERTnv_arr_binsert;
    GDB_FNC_ARR_BSEARCHnv_arr_bsearch;
    void *arr_unused[2];
    GDB_FNC_CP_REGISTER_CVTnv_cp_register_cvt;
}
```

```
GDB_FNC_CP_GET_IDnv_cp_get_id;
GDB_FNC_CP_REGISTERnv_cp_register;
GDB_FNC_DSTR_NEWnv_dstr_new;
GDB_FNC_DSTR_APPENDnv_dstr_append;
GDB_FNC_DSTR_M_APPENDnv_dstr_m_append;
GDB_FNC_DSTR_RESETnv_dstr_reset;
GDB_FNC_DSTR_LENGTHnv_dstr_length;
GDB_FNC_DSTR_INSERTnv_dstr_insert;
GDB_FNC_DSTR_CUTnv_dstr_cut;
GDB_FNC_DSTR_GET_ITEMnv_dstr_get_item;
void *dstr_unused[5];
GDB_FNC_STR_C_STRINGnv_str_c_string;
GDB_FNC_STR_C_STRING_PADnv_str_c_string_pad;
GDB_FNC_STR_STRCONCATnv_str_strconcat;
void *str_unused[5];
GDB_FNC_DT_CONVERTnv_dt_convert;
GDB_FNC_DT_ADD_DATATYPEEnv_dt_add_datatype;
void *dt_unused[5];
GDB_FNC_TRAVERSE_FILTERnv_traverse_filter;
void *filter_unused[5];
GDBH_FNC_DESCRIBE_TABLEEnv_gdbh_describe_table;
GDBH_FNC_DESCRIBE_SPnv_gdbh_describe_sp;
GDBH_FNC_GET_ITEM_LISTnv_gdbh_get_item_list;
GDB_FNC_GET_TABLE_DAnv_gdb_get_table_da;
void *gdbh_unused[5];
GDB_FNC_PROF_INIT_FILEnv_prof_init_file;
GDB_FNC_PROF_SECTION_FIRSTnv_prof_section_first;
GDB_FNC_PROF_SECTION_NEXTnv_prof_section_next;
GDB_FNC_PROF_ITEM_FIRSTnv_prof_item_first;
GDB_FNC_PROF_ITEM_NEXTnv_prof_item_next;
GDB_FNC_PROF_GET_VALUEEnv_prof_get_value;
GDB_FNC_PROF_GET_CUR_VALUEEnv_prof_get_cur_value;
GDB_FNC_PROF_SET_SECTIONnv_prof_set_section;
GDB_FNC_PROF_SET_ITEMnv_prof_set_item;
GDB_FNC_PROF_DELETEnv_prof_delete;
void *prof_unused[5];
GDB_FNC_DB_GET_PROPERTYnv_db_get_property;
GDB_FNC_DB_GET_PROPERTY_STRING nv_db_get_property_string;
GDB_FNC_DB_GET_PROPERTY_LONGnv_db_get_property_long;
GDB_FNC_DB_SET_PROPERTYnv_db_set_property;
void *db_prop_unused[5];
} GDB_HELP_FUNCTIONS;
```

**version**

The version number of the support functions that are being used. Attunity Connect checks this version to ensure compatibility between the support functions defined when you built the data driver/data type and those in the current version.

The current version number for the support functions is 110, as defined by GDB\_HELP\_VERSION\_NUMBER\_.

<b>gdb_debug_mode</b>	Use the macro GDB_DEBUG_MODE to initialize this element. If tracing is not enabled, this value is zero (0). If you enable debugging in the Attunity Connect environment settings, this value is one (1). See "Tracing the Flow of Operations in a Data Driver" on page 28.
<b>nv_memp_new</b> ... <b>nv_db_set_property</b>	See "Developer SDK Predefined Functions" on page 153 for a description of each supported function. ❖ The predefined data type macros UDT_CONVERT and UDT_ADD_DATATYPE are mapped to the corresponding functions defined in this structure (nv_dt_convert and nv_dt_add_datatype, respectively).

## ODBC\_FUNCTIONS

This structure lists all of the API functions for a generic ODBC driver. Specification of the ODBC function pointer structure (ODBC\_FUNCTIONS) is optional:

- If the ODBC driver definition (in addon.def) includes the INIT-FUNCTION keyword, Attunity Connect calls the specified entry point with a pointer to the ODBC\_FUNCTIONS structure.
- Otherwise, Attunity Connect searches the DLL to obtain the list of functions specified for the custom ODBC driver.

You can omit assignments in the structure for any functions that either are not used or do not require custom processing. If a requested function is not found in the DLL, Attunity Connect uses the corresponding standard ODBC API implemented by the ODBC driver manager on the given platform. For details, see "ODBC Support" in the *Attunity Connect Reference*.

```
typedef struct {
    ODBC_API_SQLALLOCATESQLAllocConnect;
    ODBC_API_SQLALLOCENVSQLAllocEnv;
    ODBC_API_SQLALLOCSTMTSQLAllocStmt;
    ODBC_API_SQLBINDCOLSQLBindCol;
    ODBC_API_SQLBINDPARAMETERSQLBindParameter;
    ODBC_API_SQLCOLUMNSSQLColumns;
    ODBC_API_SQLCONNECTSQLConnect;
    ODBC_API_SQLDESCRIBECKSSQLDescribeCol;
    ODBC_API_SQLDESCRIBEPARAMSQLDescribeParam;
    ODBC_API_SQLDISCONNECTSQLDisconnect;
    ODBC_API_SQLDRIVERCONNECTSQLDriverConnect;
    ODBC_API_SQLERRORSQLError;
    ODBC_API_SQLEXECDIRECTSQLExecDirect;
```

```
    ODBC_API_SQLEXECUTE SQLExecute;
    ODBC_API_SQLEXTENDEDFETCH SQLExtendedFetch;
    ODBC_API_SQLFETCHSQLFetch;
    ODBC_API_SQLFOREIGNKEYSSQLForeignKeys;
    ODBC_API_SQLFREECONNECTSQLFreeConnect;
    ODBC_API_SQLFREEENVSQLFreeEnv;
    ODBC_API_SQLFREEESTMTSQLFreeStmt;
    ODBC_API_SQLGETCONNECTOPTIONSQLOGetConnectOption;
    ODBC_API_SQLGETDATASQLGetData;
    ODBC_API_SQLGETFUNCTIONSSQLGetFunctions;
    ODBC_API_SQLGETINFO SQLGetInfo;
    ODBC_API_SQLGETTYPEINFO SQLGetTypeInfo;
    ODBC_API_SQLNUMPARAMS SQLNumParams;
    ODBC_API_SQLNUMRESULTCOLS SQLNumResultCols;
    ODBC_API_SQLPARAMDATA SQLParamData;
    ODBC_API_SQLPREPARE SQLPrepare;
    ODBC_API_SQLPRIMARYKEYSSQLPrimaryKeys;
    ODBC_API_SQLPROCEDURECOLUMNSSQLProcedureColumns;
    ODBC_API_SQLPROCURESSQLProcedures;
    ODBC_API_SQLPUTDATA SQLPutData;
    ODBC_API_SQLROWCOUNTSQLRowCount;
    ODBC_API_SQLSETCONNECTOPTIONSQLOSetConnectOption;
    ODBC_API_SQLSETSTMTOPTIONSQLOSetStmtOption;
    ODBC_API_SQLSTATISTICSSQLStatistics;
    ODBC_API_SQLTABLESSQLTables;
    ODBC_API_SQLTRANSACTSQLTransact;
} ODBC_FUNCTIONS;
```

## Data Buffer Structures

The Developer SDK defines several structures that are used primarily to pass data to and from the functions that you supply for the data driver or data type:

- GDB\_VAL\_DESC
- GDB\_ITEM\_DESC
- GDB\_FILE\_INFO
- GDB\_BUFFER
- GDB\_BUFFER\_COLUMN

### GDB\_VAL\_DESC

The GDB\_VAL\_DESC structure describes a data value. This structure is used primarily to pass data to and from custom data type functions.

```
typedef struct {
    char *data;
```

---

```

char *indicator;
long id;
unsigned long size;
unsigned long width;
long scale;
long unused1;
void *unused2;
long unused3;
} GDB_VAL_DESC;

```

<b>data</b>	The base address of the data storage.
<b>indicator</b>	The address of the null indicator byte. Set the null indicator byte to 1 if the data value being passed in the structure is null.
<b>id</b>	The column's data type, using either an Attunity Connect supported data type or a user defined data type.  The header file <b>dtypeid.h</b> lists macro definitions that uniquely identify the data types supported by Attunity Connect. For example, Attunity Connect defines the macro <b>DT_TYPE_L_</b> to identify a longword integer data type. Review the data type header file to determine the base data type for the custom data type and use the associated macro identifier as the id element.  ❖ The macro names in the data type header file differ from the corresponding ADD data types. Be sure to use the correct macro name when specifying an id element.  If the column uses a user-defined data type, specify the unique id (not the name) that you assigned in the UDT_DATATYPE structure for that data type.
<b>size</b>	The size (length) of a data element, in bytes.
<b>width</b>	The number of characters or digits (precision) for non-atomic data types.
<b>scale</b>	The scale of number. For example, $\text{value} * 10^{(\text{scale})}$ . This value should be negative.

## GDB\_ITEM\_DESC

The GDB\_ITEM\_DESC structure describes an item from the data source. The driver allocates the memory for this structure and can free it when the GET\_ITEM\_LIST function is called with a GDB\_ITEM\_LIST\_CLOSE request.

```
typedef struct {
    long unused;
    GDB_ITEM_TYPE type;
    char *name;
    char *description;
} GDB_ITEM_DESC;
```

**type**

A code indicating the type of item being described:

```
GDB_ITEM_TABLE_
GDB_ITEM_SP_
GDB_ITEM_VIEW_
GDB_ITEM_SYNONYM_
GDB_ITEM_ALIAS_
GDB_ITEM_LOCAL_TEMPORARY_
GDB_ITEM_GLOBAL_TEMPORARY_
GDB_ITEM_SYSTEM_TABLE_
```

The values in GDB\_ITEM\_TYPE are assigned using a bit-mapping mechanism. A given item may have more than one “type” assigned to it (for example, a system table in the data source has settings for GDB\_ITEM\_TABLE\_ and GDB\_ITEM\_SYSTEM\_TABLE\_). The values can be OR’d together in the GDB\_ITEM\_LIST function (see page 39), indicating that you want the returned list to include multiple types, such as system tables and views.

You can check the type settings for an item using one or more of the following macros:

```
HAS_GDB_ITEM_TABLE(name)
HAS_GDB_ITEM_SP(name)
HAS_GDB_ITEM_VIEW(name)
HAS_GDB_ITEM_SYNONYM(name)
HAS_GDB_ITEM_ALIAS(name)
HAS_GDB_ITEM_LOCAL_TEMPORARY(name)
HAS_GDB_ITEM_GLOBAL_TEMPORARY(name)
HAS_GDB_ITEM_SYSTEM_TABLE(name)
```

**name**

The name of the item. Depending on the item type, this value can be passed as input to the DESCRIBE\_TABLE function or the DESCRIBE\_SP function in order to access the item’s metadata (as allocated in the TABLE\_DA structure).

**description**

This element contains optional textual information about the item.

## **GDB\_FILE\_INFO**

The GDB\_FILE\_INFO structure describes a file associated with the driver.

---

```

typedef struct {
    char *driver_type;
    char *physical_file_name;
    int record_size;
    int file_organization;
    int record_format;
    int n_rows;
    int n_keys;
    int status;
} GDB_FILE_INFO;

```

<b>driver_type</b>	Identifies the name of the driver, as specified in the <b>addon.def</b> file. (For details, see page 20.)
<b>physical_file_name</b>	Corresponds to the FILENAME clause in ADDL syntax, which specifies the full name and location of the physical file to be described.  Some database drivers require the file suffix, while for other drivers (such as CISAM), the suffix must not be specified.
<b>record_size</b>	Corresponds to the SIZE clause in ADDL syntax, which specifies the size, in bytes of a record. This attribute is useful when you only want to use part of the record.
<b>file_organization</b>	Corresponds to the ORGANIZATION clause in ADDL syntax, which specifies the organization of a file system data provider. The organization can be: Index, Sequential or Relative. The default is Sequential.
<b>record_format</b>	Corresponds to the RECORD_FORMAT clause in ADDL syntax, which typically is specified only for non-relational data providers. The RECORD_FORMAT can be: Undefined, Fixed or Variable.
<b>n_rows</b>	Corresponds to the N_ROWS clause in ADDL syntax, which specifies the approximate number of rows in the table.
<b>n_keys</b>	The number of keys (indices) that are defined for the file.
<b>status</b>	This element returns a value indicating the results of the file lookup:  GDB_FILE_INFO_OK_ GDB_FILE_INFO_UNSUPPORTED_DRIVER_ GDB_FILE_INFO_FILE_NOT_FOUND_ GDB_FILE_INFO_UNKNOWN_ GDB_FILE_INFO_SYSTEM_ERROR_

## GDB\_BUFFER

The GDB\_BUFFER structure describes the data buffer to be built for the open stream. If the driver supports column lists, Attunity Connect builds this buffer based on the columns referenced in the data retrieval or data manipulation request. If the driver does not support column lists, Attunity Connect includes all columns in the buffer.

```
typedef struct {
    long n_columns;
    GDB_BUFFER_COLUMN *columns;
    void *buffer_data;
    long buffer_length;
    void *unused[5];
} GDB_BUFFER;
```

### n\_columns

Indicates the number of columns in the buffer.

If the driver does not support column lists Attunity Connect includes all columns in the buffer, so the driver needs to handle only the buffer\_data element.

### columns

A pointer to an array of buffer columns, as defined by the structure GDB\_BUFFER\_COLUMN.

If the driver does not support column lists Attunity Connect includes all columns in the buffer, so the driver needs to handle only the buffer\_data element.

If an array column is requested as part of a SET\_BUFFER call, the counter column for the array is requested as well.

### buffer\_data

A pointer to the actual data. If the driver does not support column lists, the driver can ignore the elements for n\_columns and columns, since Attunity Connect returns all columns in the buffer.

### buffer\_length

The length, in bytes, of the allocated buffer.

## GDB\_BUFFER\_COLUMN

The GDB\_BUFFER\_COLUMN structure describes an individual column in the data buffer.

```
typedef struct {
    long column_number;
    COLUMN_DA *column_da;
    long data_offset;
    long indicator_offset;
}
```

---

```

GDB_VAL_DESC *val_desc;
void *unused[5];
} GDB_BUFFER_COLUMN;

```

<b>column_number</b>	The ordinal number of the column in the table_da->columns array, for the field that you want to include in the buffer.
	❖ Column numbering is zero-based. For example, if you want the second column in the table to be part of the buffer, specify 1 as the column_number value for the buffer column.
<b>column_da</b>	A pointer to the column description, as defined by the structure COLUMN_DA.
<b>data_offset</b>	The offset (in bytes) within buffer_data where this field starts.
	❖ The buffer is not aligned.
<b>indicator_offset</b>	If the field is nullable, this element gives the offset (in bytes) within buffer_data where the null indicator is located. (This points to the same physical address as the ‘indicator’ element in the GDB_VAL_DESC structure.) If the field does not allow null values, the indicator_offset value is -1.
	❖ The null indicator occupies one byte in the buffer.
<b>val_desc</b>	This element points back at the buffer, viewing it as a value descriptor. This enables you to get the data type of a buffer column for which there is no metadata. For example, a stored procedure that uses SQL command syntax may not have metadata describing its parameters.

## Metadata Structures

The GDB API defines metadata structures at the following levels:

- Table (TABLE\_DA)
- Column (COLUMN\_DA)
- Index (INDEX\_DA)
- Index segment (SEG\_DA)

At each level in the metadata, the GDB API reserves space for internal use. For details, see page 283.

## TABLE\_DA

The TABLE\_DA structure, allocated by the driver, includes all the metadata Attunity Connect needs to know about a table or procedure.

```
struct TABLE_DA {
    char *table_name;
    char *table_id;
    long n_columns;
    long n_indexs;
    long n_params;
    COLUMN_DA *columns;
    INDEX_DA *indexs;
    COLUMN_DA *params;
    char *command_text;
    char *path;
    long bookmark_size;
    long cardinality;
    long attributes;
    char *description;
    char *db_command;
    GDB_COMMAND gdb_command;
    GDB_DB_FUNCTIONS *table_da_functions;
    ITABLE_DA *internal_use;
    void *unused[3];
};
```

### table\_name

Specify a name that identifies the table or procedure from this data source. This element corresponds to the table name in the DEFINE RECORD statement or DEFINE PROCEDURE statement in ADDL syntax and has the following constraints:

- The value must be unique.  
Attunity Connect uses the table name to obtain metadata details, either from ADD or from the driver (using the DESCRIBE\_TABLE and DESCRIBE\_SP routines).
- This value is case sensitive (subject to operating system requirements and the driver's GDB\_M\_CASE\_SENSITIVE\_attribute setting).
- You cannot use reserved keywords in Attunity Connect SQL for table names. See the *Attunity Connect Reference* for the list of reserved keywords.

### table\_id

The use of TABLE\_ID is data source-dependent; Attunity Connect ignores any value assigned to this field. However, Attunity

---

	recommends that you use DB_COMMAND rather than TABLE_ID to store driver-dependent information for the table.
<b>n_columns</b>	The number of columns that are exposed for the table or procedure, as listed in the subsequent COLUMNS array.
<b>n_indexs</b>	The number of indices that are exposed for the table, as listed in the indexs array (see below).
	Indexes specified for procedures are ignored.
<b>n_params</b>	For a stored procedure, this element specifies the number of input parameters in the PARAMS array that are passed to the procedure.  This element only applies to stored procedures.
<b>columns</b>	A pointer to an array of COLUMN_DA structures, listing the fields that make up a table or procedure.
<b>indexs</b>	A pointer to an array of INDEX_DA structures, listing the indices defined for the table.  This element only applies to tables.
<b>params</b>	A pointer to an array of COLUMN_DA structures, listing input parameters for the stored procedure.  This element only applies to stored procedures.
<b>command_text</b>	The command text that you want the driver to run in order to produce the rowset (table) being described.  If you specify command_text in the table description, also set the table attribute GDB_TABLE_M_COMMAND_ to identify the table as the result of a command.
<b>path</b>	Corresponds to the FILENAME clause in ADDL syntax, which specifies the full name and location of the physical file. The driver is responsible for using this value to locate and access the data (typically as part of the OPEN function).
<b>bookmark_size</b>	Bookmarks uniquely identify rows. You must supply bookmark_size whenever bookmarks are required: <ul style="list-style-type: none"><li>■ The driver allows read-write operations.</li><li>■ The driver supports arrays.</li></ul> You can set the bookmark_size when you specify the TABLE_DA structure for a given table. Alternatively, you can use the macro

GDB\_SET\_BOOKMARK\_SIZE at the driver level to set the bookmark\_size for all tables accessed by the driver. (For details, see page 259.)

**cardinality** Corresponds to the N\_ROWS clause in ADDL syntax, which specifies the approximate number of rows in the table. It is used by the Attunity Connect Query Processor.

**attributes** The GDB API supports several attributes related to tables and procedures accessed by the driver. In the TABLE\_DA structure, each individual attribute setting is mapped to a particular bit of the attributes element. The table-level attributes are listed below. (For details, see page 301.)

Attribute Name	Description
GDB_TABLE_M_STORED_PROC_	Indicates item is a stored procedure.
GDB_TABLE_M_COMMAND_	Indicates item is a command.

**description** Corresponds to the DESCRIPTION clause in ADDL syntax, which enables you to specify an optional textual description.

**db\_command** Corresponds to the DB\_COMMAND clause in ADDL syntax, which is relevant only for data drivers. The DB\_COMMAND clause enables you to specify a string containing driver-specific commands as part of the table metadata. Attunity Connect ignores any value assigned to this field.

**gdb\_command** For a command, this element identifies the cursor associated with the command\_text.

If the command results in a rowset, supply a PREPARE\_COMMAND function that returns the cursor handle for the statement. The DESCRIBE\_COMMAND function takes this cursor as input to obtain metadata for the rowset (TABLE\_DA). For details, see page 68.

**table\_db\_functions** This element points to a secondary specification of selected functions in the GDB\_DB\_FUNCTIONS structure, enabling you to supply different implementations of one or more functions for a given table. You can specify table-level implementations of the following data retrieval functions:

- OPEN (page 44)
- SET\_BUFFER (page 45)
- SET\_INDEX (page 46)
- SET\_FILTER (page 48)
- SET\_FILTER\_PARAMS (page 50)
- SET\_PARAMS (page 51)

- SET\_BM (page 51)
- FETCH (page 53)
- CLOSE (page 54)
- REWIND (page 55)
- ADD\_ROW (page 56)
- UPDATE\_ROW (page 56)
- DELETE\_ROW (page 57)
- LOCK\_ROW (page 58)
- UNLOCK\_ROW (page 58)

For all other functions in the GDB API, Attunity Connect uses the driver-level functions, regardless of any settings you make in the table metadata. Similarly, if a table-level structure element is null for one of the above functions, Attunity Connect uses the corresponding driver function.

**internal\_use**

This element is reserved by Attunity Connect for internal use and future expansion. See page 283.

## COLUMN\_DA

The COLUMN\_DA structure, allocated by the driver, includes all the metadata Attunity Connect needs to know about a column in a table/procedure or a parameter in a procedure.

```
struct COLUMN_DA {  
    char *column_name;  
    long datatype;  
    long length;  
    long scale;  
    long precision;  
    long dimension_1;  
    long dimension_2;  
    long nest_level;  
    long aux_column;  
    char *case_value;  
    char *description;  
    char *prompt;  
    char *format;  
    long attributes;  
    char *db_command;  
    long abs_offset;  
    TABLE_DA *chapter_table_da;  
    char *chapter_of;  
    ICOLUMN_DA *internal_use;
```

```
    GDB_VAL_DESC null_desc;
    void *unused[1];
}
```

**column\_name**

Specify a name that uniquely identifies this column in the table or procedure.

You cannot use reserved keywords in Attunity Connect SQL for column names. See the *Attunity Connect Reference* for the list of reserved keywords.

**datatype**

Specify the column's data type, using either an Attunity Connect supported data type or a user defined data type.

The header file **dtypeid.h** lists macro definitions that uniquely identify the data types supported by Attunity Connect. For example, Attunity Connect defines the macro **DT\_TYPE\_L\_** to identify a longword integer data type. Review the data type header file to determine whether there is a supported data type for the column, then use the associated macro identifier as this element in the **COLUMN\_DA** structure.

- ❖ The macro names in data type header file differ from the corresponding ADD data types. Be sure to use the correct macro name when specifying datatype.

If the column uses a custom data type, specify the unique id (not the name) that you assigned in the **UDT\_DATATYPE** structure.

Some columns control the behavior and use of other fields that you define. These “control” fields include columns for structures, variants, and variant cases. Control fields require special data types in their metadata definition, as follows:

Type of Control Field	Data Type Name
Structure	<b>DT_TYPE_GROUP_</b>
Variant	<b>DT_TYPE_VARIANT_</b>
Case	<b>DT_TYPE_CASE_</b>

**length**

Corresponds to the **SIZE** clause in the **DATATYPE** specification in ADDL syntax, which gives the number of characters or digits for the field.

For an atomic data type, such as **BYTE**, the length is fixed by the data type's definition. For a custom atomic data type, the length is fixed by the size element in the **UDT\_DATATYPE** structure, described on page 291.

Specify a value for non-atomic, decimal, and scaled data types. For non-atomic data types, such as strings, the length is the maximum number of bytes allocated for the field, while precision is the number of

---

	<p>bytes used for the data. For example, a null-terminated string that allows up to 10 characters of data would be specified with a length of 11 and precision of 10, to allow one byte for the null terminator.</p>
<b>scale</b>	<p>For a decimal data type, this element corresponds to the FRACTIONS clause in the DATATYPE specification in ADDL syntax, which gives the number of digits that are fractions. The number of fractions must not exceed the number of digits. The default value is 0.</p> <ul style="list-style-type: none"><li>❖ In this structure, the scale number must be negative.</li></ul>
	<p>For a scaled data type, this element corresponds to the SCALE clause in the DATATYPE specification in ADDL syntax, which gives the number of characters or digits. The number must be negative.</p>
<b>precision</b>	<p>Corresponds to the SIZE clause in the DATATYPE specification in ADDL syntax, which gives the number of characters or digits for the field.</p> <p>For atomic and scaled data types, such as BYTE and SCALED BYTE, the precision is 0 because the size is fixed by the data type's definition.</p> <p>Specify a value for non-atomic and decimal data types. For non-atomic data types, such as strings, the length is the maximum number of bytes allocated for the field, while precision is the number of bytes used for the data. For example, a null terminated string that allows up to 10 characters of data would be specified with a length of 11 and precision of 10, to allow one byte for the null terminator.</p>
<b>dimension_1</b>	<p>The maximum number of occurrences of the group of columns that make up an array column. This element applies only when a relevant data type (such as DT_TYPE_GROUP_) is specified for the selected column.</p> <p>A value of 0 indicates that the column is not an array. Since this element is zero-based, specify a value of <math>n</math> to define an array with <math>n+1</math> occurrences.</p> <ul style="list-style-type: none"><li>❖ When a STRUCTURE field has an array attribute, the field is actually an array. Each of the array elements contains all of the subordinate fields defined in this STRUCTURE field description statement. The size of the array thus is the size of a single array element multiplied by the dimension.</li></ul>
<b>dimension_2</b>	<ul style="list-style-type: none"><li>❖ <i>For Future Use:</i> This column attribute will be implemented in an upcoming version of the Developer SDK.</li></ul>
<b>nest_level</b>	<p>The grouping level for a subordinate field in a structure or variant definition. The value starts at level 1 for single columns as well as for columns defining structures (data type DT_TYPE_GROUP_) or variants</p>

(data type DT\_TYPE\_VARIANT\_). The value increases for columns that are nested within the structure or variant.

For example, a variant defined with a selector column has a nest\_level of 1. The selector columns, with a data type of DT\_TYPE\_CASE\_, have a nest\_level of 2 (along with the appropriate case\_values). Within the actual field definition, the elements of the variant have a nest\_level of 3.

**aux\_column**

For an array field, this element is the ordinal number of the column that is used as the counter for the number of occurrences of the group of columns that makes up the array.

- ❖ Column numbering is zero-based. For example, if the second column is the counter for an array field that you are defining, specify 1 for the aux\_column value in the array column's definition.

For a variant field, this element identifies the ordinal number of the optional *discriminating field*, that is, the column of relevant data that determines which of the alternate variant definitions is used in the current record (row). This corresponds to the second syntactic form of the VARIANTS field description statement in ADDL. The discriminating field definitions, with a data type of DT\_TYPE\_CASE\_ and a nest\_level of 2, define the appropriate case values. If variant cases are not specified, you should set aux\_column equal to -1.

If the column is neither an array counter nor variant field, set aux\_column equal to -1, to properly initialize the structure element.

**case\_value**

For a column that has been referenced in the aux\_column element of a variant field, this element provides an appropriate value for a variant case. (This element corresponds to the VARIANT VALUE clause in the second syntactic form of the VARIANTS field description statement in ADDL.)

**description**

Corresponds to the DESCRIPTION clause in ADDL syntax, which enables you to specify an optional textual description.

**attributes**

The GDB API supports several attributes related to columns accessed by the driver. In the COLUMN\_DA structure, each individual attribute setting is mapped to a particular bit of the attributes element. The column-level attributes are listed below. (For details, see page 301.)

Attribute Name	Description
GDB_COL_M_NULLABLE_	This attribute corresponds to the NULLABLE clause in ADDL syntax, which specifies that the field can contain a NULL value. <ul style="list-style-type: none"><li>❖ An extra byte is required to store the NULL flag (following the field in the buffer).</li></ul>

<b>Attribute Name</b>	<b>Description</b>
GDB_COL_M_NON_UPDATEABLE_	The NON_UPDATEABLE clause specifies that the field value cannot be changed once it has been stored in the table.
GDB_COL_M_BLOB_	This attribute corresponds to the BLOB (Binary Large Object) clause in ADDL syntax, which specifies that the field contains a binary image or text data. The data type for binary data is FILLER (DT_TYPE_FILLER_), and the data type for text data is STRING (DT_TYPE_T_). This clause is relevant only for files that are read by Attunity Connect.
GDB_COL_M_AUTOINCREMENT_	This element corresponds to the AUTOINCREMENT clause in ADDL syntax, which specifies that this field is updated automatically by the data source and cannot be assigned a value in an INSERT command.
GDB_COL_M_EXPLICIT_SELECT_	This element corresponds to the EXPLICIT_SELECT clause in ADDL syntax, which specifies that this field is not returned when you execute a "SELECT * FROM ..." statement.
GDB_COL_M_NULL_FLD_DESC_	This element corresponds to the NULL_VALUE clause, which specifies the null value for the field during an insert operation, when a value is not specified.
GDB_COL_M_CHAPTER_	This element indicates that the column is a chapter (child table).
GDB_COL_M_HIDDEN_	This element corresponds to the HIDDEN clause, which specifies that the column is "hidden" from the Attunity Connect Query Processor, and is never used or displayed in a query.
<b>db_command</b>	Corresponds to the DB_COMMAND clause in ADDL syntax, which is relevant only for data drivers. The DB_COMMAND clause enables you to specify a string containing driver-specific commands as part of the column metadata. Attunity Connect ignores any value assigned to this field.
<b>abs_offset</b>	Corresponds to the OFFSET IS clause in ADDL syntax, which enables you to map to specific fields in the data file, without requiring you to specify and read "filler" fields.  This element is set to zero by default, unless it is explicitly assigned a value as part of a column definition.
	To allow proper functioning of drivers created prior to Version 1.7, Attunity Connect ignores the value of abs_offset in columns if the GDB_M_SUPPORT_ABS_OFFSET_ attribute is not set for the driver.

<b>chapter_table_da</b>	Corresponds to the <i>chapter identifier</i> in the CHAPTER clause in ADDL syntax. This syntax enables you to designate a table that you want to use as a “child” rowset of the column being defined (also known as the <i>chapter column</i> ). The table identified in this element is a complete table, embedded as a single column to create a hierarchy in the “parent” table. <ul style="list-style-type: none"><li>❖ When specifying a chapter column, the data type and size information describe the “value” of the child rowset data as it is used in the context of the parent rowset.</li></ul>
<b>chapter_of</b>	The CHAPTER OF clause in ADDL syntax. A “child” table can be used independently as well as being referenced as a chapter, so this syntax enables you to identify the parent table associated with a child rowset column.
<b>internal_use</b>	Reserved by Attunity Connect for internal use. See page 283.
<b>null_desc</b>	The null value for the field during an insert operation, when a value is not specified. You must also set the column attribute GDB_COL_M_NULL_FLD_DESC_ to enable the use of this value.

## INDEX\_DA

The INDEX\_DA structure, allocated by the driver, includes all the metadata Attunity Connect needs to know about an index in the table.

```
struct INDEX_DA {  
    char *index_name;  
    char *index_id;  
    long index_length;  
    long n_segments;  
    SEG_DA *segments;  
    char *description;  
    long cardinality;  
    long attributes;  
    char *db_command;  
    IINDEX_DA *internal_use;  
    void *unused[4];  
};
```

<b>index_name</b>	A name that uniquely identifies the index in the table.
	You cannot use reserved keywords of Attunity Connect SQL for index names that are created using the CREATE_INDEX function. See the <i>Attunity Connect Reference</i> for the list of reserved keywords.

<b>index_id</b>	Corresponds to the INDEX_ID clause in ADDL syntax, which identifies the physical index for the record.  The use of INDEX_ID is data source dependent; Attunity Connect ignores any value assigned to this field. However, Attunity recommends that you use DB_COMMAND rather than INDEX_ID to store driver-dependent information for the index.
<b>index_length</b>	The sum of the length of the segments.
<b>n_segments</b>	The number of segments that make up the index.
<b>segments</b>	element is a pointer to an array of SEG_DA structures, listing the segments that make up the index.
<b>description</b>	Corresponds to the DESCRIPTION clause in ADDL syntax, which enables you to specify an optional textual description.
<b>cardinality</b>	Corresponds to the N_ROWS clause in ADDL syntax, which specifies the approximate count of distinct key values in the index.  For a unique index, the N_ROWS value must be equal to the N_ROWS value for the record (that is, the number of distinct index values is the same as the number of rows).  The index cardinality is very important for optimization. The Attunity Connect Query Processor uses the table cardinality and the index cardinality to calculate the cost of a retrieval strategy. The Query Processor divides the table cardinality by the index cardinality, and uses the result as an estimate of the number of rows that are returned if the index is used.  If this statistical information is not available in the database, set this element to 0 and later add cardinality information to the metadata using the update_statistics utility of the NAV_UTIL utility. Attunity Connect merges the cardinality information and the metadata you supply when calling the OPEN routine. (If you subsequently run the add_to_addl utility, you see the N_ROWS clauses with the cardinality values.) <ul style="list-style-type: none"><li>❖ Specify extended metadata for the custom data source in order to use the update_statistics option. See the <i>Attunity Connect Reference</i> for details.</li></ul>
<b>attributes</b>	The GDB API supports several attributes related to indices accessed by the driver. In the INDEX_DA structure, each individual attribute

setting is mapped to a particular bit of the attributes element. The index-level attributes are listed below. (For details, see page 302.)

Attribute Name	Description
GDB_INDEX_M_HASHED_	This attribute corresponds to the HASHED clause in ADDL syntax, which indicates that this is a hash index. This optional clause is relevant only for the query optimization strategies used by Attunity Connect.
GDB_INDEX_M_UNIQUE_	This element corresponds to the UNIQUE clause in ADDL syntax, which indicates that each index entry uniquely identifies one row. This optional clause is relevant only for the query optimization strategies used by Attunity Connect.
GDB_INDEX_M_HIERARCHY_	This type of index tells the Attunity Connect Query Processor that, if there is a possible strategy using this index, it should use the given strategy rather than any other query optimization.
GDB_INDEX_M_CLUSTERED_	This element corresponds to the CLUSTERED clause in ADDL syntax, which indicates that this index reflects the physical organization of the table. This optional clause is relevant only for the query optimization strategies used by Attunity Connect.
GDB_INDEX_M_BEST_UNIQUE_	This attribute tells the Attunity Connect Query Processor that a strategy using this index has a lower cost than any other query optimization strategy used by Attunity Connect, including other unique indices for the underlying table.
<b>db_command</b>	Corresponds to the DB_COMMAND clause in ADDL syntax, which is relevant only for data drivers. The DB_COMMAND clause enables you to specify a string containing driver-specific commands as part of the index metadata. Attunity Connect ignores any value assigned to this field.
<b>internal_use</b>	This element is reserved by Attunity Connect for internal use. See page 283.

## SEG\_DA

The SEG\_DA structure, allocated by the driver, includes all the metadata Attunity Connect needs to know about an index segment.

```
struct SEG_DA {  
    long column_number;  
    long attributes;  
    char *db_command;
```

```

long cardinality;
ISEG_DA *internal_use;
void *unused[3];
} ;

```

**column\_number** The ordinal number of the column in the table/procedure's COLUMNS array, for the field that you want to include in the index.

- ❖ Column numbering is zero-based. For example, if the second column in the table is the index segment that you are defining, specify 1 as the column\_number value for the segment.

**attributes** The GDB API supports one attribute related to index segments accessed by the driver. In the SEG\_DA structure, this individual attribute setting is mapped to a particular bit of the attributes element. The index segment-level attribute is listed below. (For details, see page 303.)

Attribute Name	Description
GDB_SEG_M_DESCENDING_	This attribute corresponds to the ORDER clause in ADDL syntax, which specifies whether the order of the current index segment is ascending or descending. For the GDB API, an index segment is assumed to be ascending unless you explicitly set this attribute.
db_command	Corresponds to the DB_COMMAND clause in ADDL syntax, which is relevant only for data drivers. The DB_COMMAND clause enables you to specify a string containing driver-specific commands as part of the index segment metadata. Attunity Connect ignores any value assigned to this field.
cardinality	<ul style="list-style-type: none"> <li>❖ <i>For Future Use:</i> This index segment attribute will be implemented in an upcoming version of the Developer SDK.</li> </ul>
internal_use	This element is reserved by Attunity Connect for internal use. See "Attunity Connect Internal Structures", below.

## Attunity Connect Internal Structures

At each level in the metadata structures, Attunity Connect allocates space in the structure for its internal use. The following structure definitions are used to allocate this space (at the level indicated by the name):

```

typedef void* ITABLE_DA;
typedef void* ICOLUMN_DA;

```

```
typedef void* IINDEX_DA;
typedef void* ISEG_DA;
```

## Filter Structure

### GDB\_FTREE\_NODE

The filter tree array uses the GDB\_FTREE\_NODE structure to specify the a pair of elements for each item in the hierarchy:

```
typedef struct {
    GDB_TREENODE treenode_type;
    long column_number;
    long param_number;
    char *const_string;
    GDB_FTREE_NODE_HANDLE subtree_node_handle;
} GDB_FTREE_NODE;
```

#### treenode\_type

A numeric code indicating the type of item that you are describing (column, operator, constant, parameter), using the GDB\_TREENODE enumeration:

```
typedef enum {
    GDB_TREENODE_COLUMN_ = 0,
    GDB_TREENODE_OPERATOR_,
    GDB_TREENODE_CONST_,
    GDB_TREENODE_PARAMETER_,
    GDB_TREENODE_SUBTREE_
#ifndef _VAR_ENUM_MACHINE
    ,GDB_TREENODE_LAST_VALUE = 0x7FFFFFFF
#endif
} GDB_TREENODE;

❖ The parameter array must be set using the function SET_FILTER_PARAMS.  
(See page 50.)
```

#### column\_number / param\_number / const\_string

The second element in the pair is the actual data for the item. The value you supply is based on the type of item, that is, either the ordinal number (in their respective arrays) of the column, parameter, or constant, or parameter, or a numeric code indicating the type of operator. The following operator types can appear in filter expressions, using the GDB\_FTREE\_OPERATOR enumeration:

```
typedef enum {
    TREEOP_LT_ = 0,
    TREEOP_LE_,
    TREEOP_EQ_,
```

```

TREEOP_GE_,
TREEOP_GT_,
TREEOP_LIKE_,
TREEOP_LIKE3_, /* LIKE, with escape character */
TREEOP_BETWEEN_, /* usually received as GE and LE */
TREEOP_AND_,
TREEOP_OR_,
TREEOP_ISNULL_, /* only 1 operand for this value */
TREEOP_NOT_, /* only 1 operand for this value */
#ifndef _VAR_ENUM_MACHINE
    ,GDB_FTREE_OPERATOR_LAST_VALUE = 0x7FFFFFFF
#endif
} GDB_FTREE_OPERATOR;

```

- ❖ Note that whenever a LIKE operator is included in a query the query processor passes a filter which includes the LIKE operator and an equivalent expression without the LIKE operator. For example, a clause such as:  
B LIKE 'ABC%'  
is passed by the query processor as:  
B LIKE 'ABC%' AND B>= 'ABC' AND B < 'ABD'.

**Example**

Assuming a table with columns named Col1, Col2, Col3, Col4, and Col5, the following filter expression is valid:

((Col5=Const1) AND (Col2<Col3)) OR (Col1=Param1)

The logical array representing this filter as a tree is:

```

OR
AND
EQ
Col5
Const1
LE
Col2
Col3
EQ
Col1
Param1

```

The corresponding function arguments for SET\_FILTER (see page 48) are:

n\_filter\_tree = 22

filter\_tree (array):

Value	Description
1	TREENODE_OPERATOR_
8	TREEOP_OR_
1	TREENODE_OPERATOR_
7	TREEOP_AND_
1	TREENODE_OPERATOR_
2	TREEOP_EQ_
0	TREENODE_COLUMN_
4	Ordinal of Col5 (zero-based)
2	TREENODE_CONST_
0	Ordinal of Const1 entry in filter_consts array (zero-based)
1	TREENODE_OPERATOR_
0	TREEOP_LT_
0	TREENODE_COLUMN_
1	Ordinal of Col2 (zero-based)
0	TREENODE_COLUMN_
2	Ordinal of Col3 (zero-based)
1	TREENODE_OPERATOR_
2	TREEOP_EQ_
0	TREENODE_COLUMN_
0	Ordinal of Col1 (zero-based)
3	TREENODE_PARAMETER_
0	Ordinal of Param1 entry in filter_params array (zero-based)

n\_filter\_consts = 1

filter\_consts (array):

<string value for Const1>

- ❖ The parameter array is set using the function SET\_FILTER\_PARAMS. (See page 50.) For this example, n\_params = 1 and filter\_params array contains the value for 'Param1'.

### Ignoring Driver Operators

There is no explicit way to determine which operators are supported by a driver. You can, however, use code like the following to ignore an operator:

```
static STRING operator_expression(GDB_FTREE_OPERATOR node_operator)
{
    static STRING op_text[10] =
    { "<", "<=", "==" , ">=" , ">" , "", "", "", "and", "or" } ;
```

```
        return (op_text[node_operator]) ;
}

static STRING operand_expression(GDB_FTREE_NODE *operand,
                                TABLE_DA *table_da,
                                void *memory)
{
    STRING col_name ;
    STRING const_value ;

    switch (operand->treenode_type) {
    case GDB_TREENODE_COLUMN_:
        DSTR_NEW(&col_name, "", 0, 64, memory) ;
        DSTR_M_APPEND(&col_name, "x.",
                      table_da->columns[operand->column_number].column_name,
                      NULL) ;
        return (col_name) ;

    case GDB_TREENODE_CONST_:
        DSTR_NEW(&const_value, "", 0, 64, memory) ;
        DSTR_M_APPEND(&const_value,
                      "\"", operand->const_string, "\"",
                      NULL) ;
        return (const_value) ;

    case GDB_TREENODE_PARAMETER_:
        return ("?") ;

    case GDB_TREENODE_SUBTREE_:
    case GDB_TREENODE_OPERATOR_:
        return (operand->subtree_node_handle) ;

    default:
        return (" ??Unknown operand?? ") ;
    }
}

static STRING merge_unsupported_node(GDB_FTREE_OPERATOR node_operator,
                                    GDB_FTREE_NODE *other_operand)
{
    switch (node_operator) {
    case TREEOP_OR_:
        return (NULL) ;

    case TREEOP_AND_:
        /* Doesn't make sense to AND with anything other than a subtree */

```

```
        if (other_operand->treenode_type != GDB_TREENODE_SUBTREE_)
            return (NULL) ;
        else
            return (other_operand->subtree_node_handle) ;

/* Doesn't make sense to have a subtree with a different operator */
default:
    return (NULL) ;
}
}

static GDB_STATUS process_ftree_node(
    GDB_FTREE_HANDLE io_stream,
    GDB_FTREE_NODE_HANDLE *node_handle,
    GDB_FTREE_OPERATOR node_operator,
    long n_node_operands,
    GDB_FTREE_NODE **node_operands)
{
    XSTREAM *XSTREAM = io_stream ;
    XCONNECT *XCONNECT = XSTREAM->XCONNECT ;
    TABLE_DA *table_da = XSTREAM->table_da ;
    void *memory = XSTREAM->tmp_memory ;
    STRING subtree_expression ;
    long i ;

    /* Check if operator is supported */
    switch (node_operator) {
    case TREOP_LT_:
    case TREOP_LE_:
    case TREOP_EQ_:
    case TREOP_GE_:
    case TREOP_GT_:
    case TREOP_AND_:
    case TREOP_OR_:
        break ;

    default:
        *node_handle = NULL ;
        return (GDB_OK_) ;
    }

    /* loop on operands. All supported operators are binary */
    for (i=0; i<2; i++) {
        GDB_FTREE_NODE *operand = node_operands[i] ;
        GDB_FTREE_NODE *other_operand = node_operands[1-i] ;

        /* Check if operand is an unsupported subtree */
```

```

        if (operand->treenode_type == GDB_TREENODE_SUBTREE_ &&
            !operand->subtree_node_handle) {
            node_handle = merge_unsupported_node(node_operator,
other_operand) ;
            return (GDB_OK_) ;
        }

        if (operand->treenode_type == GDB_TREENODE_COLUMN_) {
            COLUMN_DA *column_da =
&table_da->columns[operand->column_number] ;
            char *prop_type = DSTR_GET_ITEM(column_da->db_command,
JCOL_PROP_TYPE_, '/', memory) ;

            /* Ignore virtual columns in the filter tree */
            if (!strcmp(prop_type, "VIRTUAL")) {
                *node_handle = NULL ;
                return (GDB_OK_) ;
            }
        }
    }

/* Compose sub-expression for this node as "({op1} {operator} {op2})" */
DSTR_NEW(&subtree_expression, "", 0, 64, memory) ;
DSTR_M_APPEND(&subtree_expression,
"(", operand_expression(node_operands[0], table_da, memory),
" ", operator_expression(node_operator), " ",
operand_expression(node_operands[1], table_da, memory), ")",
NULL) ;

/* Return the sub-expression as the node handle */
*node_handle = subtree_expression ;
return (GDB_OK_) ;
}

static GDB_STATUS db_xxx_set_filter(
    GDB_STREAM io_stream,
    long n_filter_tree,
    long *filter_tree,
    long n_filter_consts,
    GDB_VAL_DESC **filter_consts)
{
    XSTREAM *XSTREAM = io_stream ;
    XCONNECT *XCONNECT = XSTREAM->XCONNECT ;
    TABLE_DA *table_da = XSTREAM->table_da ;
    void *memory = XSTREAM->memory ;
    STRING where_text ;
}

```

```
GDB_STATUS status ;

MEMP_FREE_POOL(&XSTREAM->tmp_memory) ;

/* Traverse filter tree and call process_ftree_node for each node */
status = GDB_TRAVERSE_FILTER(XSTREAM, &where_text,
    n_filter_tree, filter_tree,
    n_filter_consts, filter_consts,
    process_ftree_node) ;
RET_IF_ERROR(status) ;

DSTR_RESET(XSTREAM->where_clause) ;

/* where_text is returned as node_handle of root node in the tree */
if (where_text && *where_text)
    DSTR_M_APPEND(&XSTREAM->where_clause, "\nwhere ",
where_text, NULL) ;

return (GDB_OK_) ;
}
```

## Data Type Structure

### UDT\_DATATYPE

The UDT API includes a structure definition for the UDT\_DATATYPE structure. You must populate this structure with various data type attributes and the names of functions that implement the UDT API for the data type.

- ❖ You must assign the structure members prior to registering the data type with the UDT\_ADD\_DATATYPE macro.

```
typedef struct {
    long version;
    long id;
    char *name;
    char *description;
    long size;
    long base_id;
    long attributes;
    UDT_FNC_TO_STRING to_string;
    UDT_FNC_FROM_STRING from_string;
    UDT_FNC_TO_BASE to_base;
    UDT_FNC_FROM_BASE from_base;
    UDT_FNC_COMPARE compare;
    UDT_FNC_GET_LAST_ERROR get_last_error;
```

```

UDT_FNC_IS_EMPTY_VAL is_empty_val;
UDT_FNC_SET_EMPTY_VAL set_empty_val;
UDT_FNC_IS_NULL_VAL is_null_val;
UDT_FNC_SET_NULL_VAL set_null_val;
UDT_FNC_CONVERT_TO convert_to;
UDT_FNC_LOWEST_VAL lowest_val;
UDT_FNC_HIGHEST_VAL highest_val;
UDT_FNC_RANDOM_VAL random_val;
UDT_FNC_DEFAULT_FORMAT default_format;
UDT_FNC_TO_EXPOSED to_exposed;
UDT_FNC_WIDTH_TO_SIZE width_to_size;
long *convert_to_datatypes;
long max_display_length;
void *unused[10];
} UDT_DATATYPE;

```

<b>version</b>	Specify the version number of the GDB API that is used with this custom data type. Attunity Connect checks this version to ensure compatibility between the structures defined in the GDB API version when you established the data type definition and the current version of the GDB API.
	Use the macro GDB_INITIALIZE in the data type's INIT-FUNCTION (registration function) to set this element equal to GDB_VERSION_NUMBER_, which is defined in the header file for the GDB API.
<b>id</b>	For each data type that you define, you must assign a unique numeric code. In the current version, the range of allowed data type codes is between 200 and 250.  Attunity Connect reserves data type codes from 0 through 199.
<b>name</b>	A name that uniquely identifies the custom data type.  You cannot use any of the macro names that Attunity Connect defines for the standard data types in the header file <b>dtypeid.h</b> .
<b>description</b>	This element enables you to specify an optional textual description for the custom data type, similar to the DESCRIPTION clause for a field in ADDL syntax.
<b>size</b>	For standard atomic data types, such as BYTE, the length is fixed by the data type's definition. When you define a custom atomic data type, specify the size that Attunity Connect uses as the maximum number of bytes that can be allocated for a field of this data type. This value is the actual number of bytes physically written to disk for a field of this data type.

You do not need to specify size for custom non-atomic, decimal, and scaled data types. In these cases, the actual value is set as ‘length’ and ‘precision’ when a field using this data type is defined in the COLUMN\_DA structure.

**base\_id**

Each custom data type must have a “base” data type (usually the closest data type that Attunity Connect supports).

- ❖ You cannot define an atomic data type with a non-atomic base data type.

The header file **dtypeid.h** lists macro definitions that uniquely identify the data types supported by Attunity Connect. For example, Attunity Connect defines the macro DT\_TYPE\_L\_ to identify a longword integer data type. Review the data type header file to determine the base data type for the custom data type and use the associated macro identifier as the **base\_id** in the UDT\_DATATYPE structure.

- ❖ The macro names in the data type header file differ from the corresponding ADD data types. Be sure to use the correct macro name when specifying a **base\_id**.

If the column is based on another custom data type, specify the unique id (not the name) that you assigned in the UDT\_DATATYPE structure for that data type.

For more information about data type support in Attunity Connect, see “ADD Supported Data Types” in the *Attunity Connect Reference*.

**attributes**

The UDT API supports several attributes related to the use of a custom data type. In the UDT\_DATATYPE structure, each individual attribute setting is mapped to a particular bit of the attributes element. The data type-level attributes are listed below. (For details, see page 303.)

Attribute Name	Description
UDT_M_DATE_	Indicates whether the data type stores date information.
UDT_M_NUMBER_	Indicates whether the data type stores numeric information.
UDT_M_SCALED_	Indicates whether the data type includes a scale value.
UDT_M_NULLABLE_	For integer data types (such as int, uint, quad), indicates whether the data type allows a null value.
UDT_M_SIZED_	Indicates whether the data type requires size information in addition to width (length).
UDT_M_SCALED_SIZE_	Indicates whether the size for the data type depends on both width (length) and scale.
UDT_M_FIXED_WIDTH_	Indicates whether the data type has a fixed width, regardless of the actual data value.
<b>to_string</b>	The name of the data type function that corresponds to the TO_STRING routine in the UDT API.

---

	This element is required.
<b>from_string</b>	The name of the data type function that corresponds to the FROM_STRING routine in the UDT API.
	This element is required.
<b>to_base</b>	The name of the data type function that corresponds to the TO_BASE routine in the UDT API.
<b>from_base</b>	The name of the data type function that corresponds to the FROM_BASE routine in the UDT API.
<b>compare</b>	The name of the data type function that corresponds to the COMPARE routine in the UDT API.
<b>get_last_error</b>	The name of the data type function that corresponds to the GET_LAST_ERROR routine in the UDT API.
<b>is_empty_val</b>	The name of the data type function that corresponds to the IS_EMPTY_VAL routine in the UDT API.
<b>set_empty_val</b>	The name of the data type function that corresponds to the SET_EMPTY_VAL routine in the UDT API.
<b>is_null_val</b>	The name of the data type function that corresponds to the IS_NULL_VAL routine in the UDT API.
<b>set_null_val</b>	The name of the data type function that corresponds to the SET_NULL_VAL routine in the UDT API.
<b>convert_to</b>	The name of the data type function that corresponds to the CONVERT_TO routine in the UDT API.
<b>lowest_val</b>	The name of the data type function that corresponds to the LOWEST_VAL routine in the UDT API.
<b>highest_val</b>	The name of the data type function that corresponds to the HIGHEST_VAL routine in the UDT API.
<b>random_val</b>	The name of the data type function that corresponds to the RANDOM_VAL routine in the UDT API.
<b>default_format</b>	The name of the data type function that corresponds to the DEFAULT_FORMAT routine in the UDT API.

<b>to_exposed</b>	The name of the data type function that corresponds to the TO_EXPOSED routine in the UDT API.
<b>width_to_size</b>	The name of the data type function that corresponds to the WIDTH_TO_SIZE routine in the UDT API.
<b>max_display_length</b>	The maximum display length of a field either with an atomic data type (the UDT_DATATYPE member is specified with a non-zero value) or with a fixed width data type (the SET_FIXED_WIDTH function is specified), used when converting a value of UDT_DATATYPE into a string.
<b>convert_to_datatypes</b>	Supply a list of data type ids for which the CONVERT_TO function should be used. (For details, see page 136.) Terminate the list with the value defined as UDT_END_CONVERT_LIST (in the current version, -1).  Before converting data of this data type to another data type, Attunity Connect first checks whether the target data type is listed in the convert_to_datatypes, and uses the CONVERT_TO function if possible. Otherwise, Attunity Connect attempts to use the TO_BASE or TO_STRING functions for the data type to do the conversion.

## Attributes

Several structures defined in the GDB API and the UDT API use a bit-mapped ‘attributes’ element to provide support for various properties that you may need in a data driver or data type. This section describes each attribute in detail, divided into categories according to the structures in which they appear.

The GDB API and the UDT API also supply macros to set and check the attributes, as listed in each attribute description.

- ❖ Not all combinations of attributes are legal when defining a custom data type. For example, a data type can be defined as a scaled numeric, but not as a scaled numeric date.

## Driver Attributes (GDB\_DB\_FUNCTIONS)

**GDB\_M\_ADD\_** Indicates whether driver metadata is stored in the Attunity Connect Attunity Connect Data Dictionary (ADD).

### Notes

- When using metadata from the Attunity Connect data Dictionary (ADD), each table or stored procedure requires a separate ADD metadata file. Attunity Connect includes a GUI-based editor and

several utility programs that enable you to create and update ADD metadata, as described in the *Attunity Connect Reference*.

- For a driver that uses metadata from the Attunity Connect ADD, the first part of the connect string must identify the ADD directory, followed by a comma and the actual connect information required to access the data source.
- If you do not use Attunity Connect ADD for metadata, you must supply additional driver routines that Attunity Connect calls to obtain the necessary metadata information when you access data using this driver. See "Defining Metadata" on page 38.

#### **GDB\_M\_SUPPORT\_COLUMN\_LIST**

Indicates whether the driver can perform read and update operations based on a requested list of columns, or only on entire records.

##### **Notes**

- By default, if a driver is designed to operate with the backend database in units of whole records, Attunity Connect assumes the responsibility for handling specific lists of columns in both read and update operations. Attunity Connect always sends complete records in the buffer to the driver in update operations (including insertions), and expects the driver to supply complete records in read operations. Attunity Connect subsequently filters the data to present only the selected columns identified in a user's request.
- If the driver can perform operations at the column level, setting the GDB\_M\_SUPPORT\_COLUMN\_LIST\_ attribute enables Attunity Connect to process requests more efficiently. In this case, Attunity Connect requests specific columns for read operations and supplies only specific columns for update.

#### **GDB\_M\_INSERT\_ALL\_COLUMNS**

For drivers that support column lists, this attribute indicates whether the driver requires a complete record buffer of all columns when inserting a new row, regardless of the driver's ability to work at the column level for other operations.

##### **Notes**

- If this attribute is assigned, Attunity Connect creates a full buffer with all columns when adding a record for the table. If values are

not supplied for all columns prior to the insert, “empty” values are used as the default.

- If the driver does not support column lists, the setting of this attribute is irrelevant; Attunity Connect always creates a complete buffer when adding a new record.

#### **GDB\_M\_SUPPORT\_TRANS\_**

Indicates whether the driver supports transactions.

##### **Notes**

If you set this attribute, you must supply driver routines for starting, aborting, and committing transactions.

#### **GDB\_M\_SUPPORT\_UPDATE\_**

Indicates whether the driver allows update operations.

##### **Notes**

- If not specified, Attunity Connect supports read-only access by the driver to application data.
- If you set this attribute, you must supply driver routines for adding, deleting, updating, locking, and unlocking rows.
  - ❖ Supply a bookmark and related driver routines if the driver supports update operations. Bookmarks may also be required in cases when the driver does not support update (for example, when blobs are used).
- You can use the data definition routines (CREATE\_TABLE, DROP\_TABLE, CREATE\_INDEX) regardless of the setting of the GDB\_M\_SUPPORT\_UPDATE\_attribute.

#### **GDB\_M\_SUPPORT\_MULTI\_THREAD\_**

Indicates whether the driver is multi-thread-safe (whether it can handle separate threads for multiple clients accessing a shared server process at the same time).

##### **Notes**

- This is an issue with multi-client multi-threaded servers, which are supported only on NT platforms in the current version of Attunity Connect. On such platforms, if the driver does not support multi-threading, Attunity Connect protects the driver so that only one thread accesses it at a time. Attunity Connect is responsible for synchronizing the client requests above the driver level. This

impedes performance, however, since other client requests wait until the driver is available.

- ❖ Even though the threads are synchronized, there is no guarantee that the driver is called from the same thread. See "GDB\_M\_DISABLE\_GST\_" on page 297 for details about how to resolve this issue.

#### **GDB\_M\_SUPPORT\_PASSTHRU\_COMMAND\_**

Indicates whether the driver supports PASSTHRU command syntax.

- ❖ *Upgrade Note:* This attribute replaces the attribute formerly named GDB\_M\_SUPPORT\_COMMAND, and uses the same mapped bit location.

#### **Notes**

- If the driver fully supports SQL, you do not need to support PASSTHRU command syntax. For details, see page 300.
- If this attribute is assigned, the Query Processor does not generate any SQL for commands associated with the driver, but rather, simply pass the command text intact to the driver.
- If this attribute is not assigned, Attunity Connect issues an error whenever a user tries to specify PASSTHRU syntax.

#### **GDB\_M\_CASE\_SENSITIVE\_**

Indicates whether the driver is case sensitive with regard to application data and metadata names (column names, table names, and so on).

- ❖ Support for case sensitivity is subject to operating system requirements.

#### **Notes**

- By default, if the driver uses Attunity Connect Data Dictionary (ADD) for metadata, the names defined in the ADD must conform to Attunity Connect ADD metadata format (for example, table names are case sensitive in ADD syntax while column names are not). You can override this default by setting the GDB\_M\_CASE\_SENSITIVE\_ attribute for the driver.

#### **GDB\_M\_DISABLE\_GST\_**

Indicates whether the driver can be called from different threads or only from the main thread.

#### **Notes**

- Query execution makes heavy use of multi-threading. A query plan (generated by the Attunity Connect query optimizer) is a tree, with leaf nodes each representing a particular table or data stream. Each leaf node has a read-ahead cache associated with it, which is

populated by a separate General Service Thread (GST). Thus much of the query processing occurs in parallel. The root node of the query tree – representing the output of the query – also has an asynchronous read-ahead cache, which is populated in parallel with user processing. This is especially effective on client machines (Windows 95/98 and NT platforms) when the backend data is on a remote server machine – the client application processing and server query processing tend to overlap almost entirely.

- The GDB\_M\_DISABLE\_GST\_ attribute complements the GDB\_M\_SUPPORT\_MULTI\_THREAD\_ attribute that controls parallel access to the driver. Even if a driver does not support multi-threading, it might be called synchronously from different threads, which may cause problems for some drivers. By disabling the GST, you ensure that the driver is always called from the main thread.

For example, the supplied Attunity Connect drivers for Sybase and OLE DB operate with the GST disabled.

#### **GDB\_M\_SUPPORT\_FILTERS\_**

Indicates whether the driver supports filtering expressions that do not necessarily apply to an index being used to access data.

##### **Notes**

- If you set this attribute, you must supply a function to accept the filter expression. Attunity Connect provides a help function TRAVERSE\_FILTER that you can use to set up the filter tree array in the format required by Attunity Connect. See the definition of the structure GDB\_FTREE\_NODE for more details.

#### **GDB\_M\_SUPPORT\_FILTER\_PARAMS\_**

Indicates whether the driver supports parameters passed in filter expressions.

##### **Notes**

- In many cases, a filter expression can be reused repeatedly with only a change in parameter values, using the SET\_FILTER\_PARAMS function. This is common when the table is participating in a join with other tables and the parameter values are set according to data items read from the other tables. If the backend database supports parameters, taking advantage of this can greatly improve performance.
- If the backend database supports filter expressions but restricts the use of parameters, do not set this attribute. In this case, Attunity

Connect converts parameters to constants and then call the driver using the SET\_FILTER function (which includes constants instead of parameters).

The setting of this attribute is ignored if GDB\_M\_SUPPORT\_FILTERS\_ is not set for the driver.

#### **GDB\_M\_FILTER\_WO\_INDEX\_**

Indicates whether the driver uses only information from the filter expression and not an index, to filter data.

##### **Notes**

- Normally, the filter tree ignores filter expressions for columns in the index that was selected by the query processor, because it assumes index processing is more efficient than filtering the entire expression. To override the default behavior, you can set this attribute and then the criteria on the selected index columns is part of the filter tree. Attunity Connect still calls the SET\_INDEX function to communicate the selected index number to the driver, but always sets zero segments in the index buffer. The processing of the entire filter expression occurs in the SET\_FILTER function.
- The setting of this attribute is ignored if GDB\_M\_SUPPORT\_FILTERS\_ is not set for the driver.

#### **GDB\_M\_FILTER\_WO\_SEGMENTS\_**

Indicates whether the driver requires separate filtering of data that would return segments of an index.

##### **Note**

- Assuming a filter expression such as:  
 $(\text{EMPID} = 5 \text{ OR } \text{EMPID} = 7) \text{ AND } \text{DATE} > 1-\text{JAN}-1998$   
the resultant data would access two continuous ranges in an index based on the EMPID column. If FILTER\_WO\_SEGMENTS is set, Attunity Connect issues a separate call to SET\_INDEX for each

value of EMPID, followed by a call to SET\_FILTER with any remaining filter information (in this case, DATE > 1-JAN-1998).

- The setting of this attribute is ignored if GDB\_M\_SUPPORT\_FILTERS\_ is not set for the driver.

#### **GDB\_M\_SUPPORT\_EXTENDED\_NAMES\_**

The driver uses Attunity Connect extended metadata to support virtual tables for arrays, regardless of whether the driver uses ADD for other metadata.

##### **Notes**

- To use extended metadata for a non-ADD data driver, specify a TDP\_EXTENDED\_DIR parameter when you identify data sources in the Attunity Connect binding file. Attunity Connect generates metadata for virtual tables into the designated directory and merges this information with metadata supplied by the driver.

#### **GDB\_M\_SUPPORT\_SQL\_**

Indicates whether the driver supports SQL command syntax.

##### **Notes**

- Set this attribute for a relational driver, so that the Attunity Connect Query Processor generates appropriate SQL syntax for commands that you supply as part of a table definition. The type of SQL syntax is defined in the GDB\_DB\_FUNCTIONS structure ("sql\_syntax\_name" on page 259).
- If the driver fully supports SQL, you do not need to support PASSTHRU command syntax described on page 297.

#### **GDB\_M\_SUPPORT\_ABS\_OFFSET\_**

Indicates whether the driver supports the use of absolute offset, defined as an element in the COLUMN\_DA structure.

##### **Notes**

- To enable proper functioning of drivers created prior to Version 1.7, Attunity Connect ignores the value of "abs\_offset" on page 279

unless the GDB\_M\_SUPPORT\_ABS\_OFFSET\_ attribute is set for the driver.

## Table Attributes (TABLE\_DA)

### GDB\_TABLE\_M\_STORED\_PROC\_

The table description (TABLE\_DA) structure is the result of a stored procedure.

### GDB\_TABLE\_M\_COMMAND\_

The table description (TABLE\_DA) structure is the result of a command.

## Column Attributes (COLUMN\_DA)

### GDB\_COL\_M\_NULLABLE\_

This attribute corresponds to the NULLABLE clause in ADDL syntax, which specifies that the field can contain a NULL value.

### GDB\_COL\_M\_NON\_UPDATEABLE\_

The NON\_UPDATEABLE clause specifies that the field value cannot be changed once it has been stored in the table.

### GDB\_COL\_M\_BLOB\_

This attribute corresponds to the BLOB (Binary Large Object) clause in ADDL syntax, which specifies that the field contains a binary image or text data. The data type for binary data is FILLER (DT\_TYPE\_FILLER\_), and the data type for text data is STRING (DT\_TYPE\_T\_). This clause is relevant only for files that are read by Attunity Connect.

#### Notes

- If you set this attribute, you must supply driver routines for BLOB processing.

### GDB\_COL\_M\_AUTOINCREMENT\_

This attribute corresponds to the AUTOINCREMENT clause in ADDL syntax, which specifies that this field is updated automatically by the data source and cannot be assigned a value in an INSERT command.

### GDB\_COL\_M\_EXPLICIT\_SELECT\_

This attribute corresponds to the EXPLICIT\_SELECT clause in ADDL syntax, which specifies that this field is not returned when you execute a “SELECT \* FROM ...” statement.

To return this field, you must explicitly ask for it. For example, in a query such as "SELECT \*, SYSKEY FROM NATION" where SYSKEY is a field defined with EXPLICIT\_SELECT.

#### **GDB\_COL\_M\_NULL\_FLD\_DESC\_**

This attribute corresponds to the NULL\_VALUE clause, which specifies the null value for the field during an insert operation, when a value is not specified. You must specify the null value for the field using the **null\_desc** member of the "COLUMN\_DA" structure.

#### **GDB\_COL\_M\_CHAPTER\_**

The column is a chapter (child table).

#### **GDB\_COL\_M\_HIDDEN\_**

This element corresponds to the HIDDEN clause, which specifies that the column is "hidden" from the Attunity Connect Query Processor and is never used or displayed in a query.

### **Index Attributes (INDEX\_DA)**

#### **GDB\_INDEX\_M\_HASHED\_**

This attribute corresponds to the HASHED clause in ADDL syntax, which indicates that this is a hash index. This optional clause is relevant only for the query optimization strategies used by Attunity Connect.

#### **Notes**

- The Attunity Connect Query Processor does not assume that the driver returns the rows sorted by the key values or in ascending/descending order. Thus, the Query Processor uses this index only in an EQUAL (=) type of relationship. The Query Processor does not use a hashed index if the query contains a greater than (>) or less than (<) operator. For example, "... where age > 50 ..." and age has a hashed index on it.

#### **GDB\_INDEX\_M\_UNIQUE\_**

This attribute corresponds to the UNIQUE clause in ADDL syntax, which indicates that each index entry uniquely identifies one row. This optional clause is relevant only for the query optimization strategies used by Attunity Connect.

### Notes

- The FETCH function is called only once for a GDB\_EQUAL\_ type of call. The cardinality might also be inferred from the table's cardinality.

#### GDB\_INDEX\_M\_HIERARCHY\_

This type of index tells the Attunity Connect Query Processor that, if there is a possible strategy using this index, it should use the given strategy rather than any other query optimization.

### Notes

- This attribute enables you to efficiently process hierarchical type of data sources, where the data definition includes virtual columns with virtual indices that represent the hierarchy. For example, Attunity Connect uses hierarchical indices in the DBMS driver and to process virtual array tables.

#### GDB\_INDEX\_M\_CLUSTERED\_

This element corresponds to the CLUSTERED clause in ADDL syntax, which indicates that this index reflects the physical organization of the table. This optional clause is relevant only for the query optimization strategies used by Attunity Connect.

#### GDB\_INDEX\_M\_BEST\_UNIQUE\_

This attribute tells the Attunity Connect Query Processor that a strategy using this index has a lower cost than any other query optimization strategy used by Attunity Connect, including other unique indices for the underlying table. This setting is intended primarily for a key that provides some kind of direct access to the record, such as the DBKEY in DBMS, the RFA in RMS (*future use*), and the ISN in ADABAS.

## Index Segment Attributes (SEG\_DA)

#### GDB\_SEG\_M\_DESCENDING\_

This attribute corresponds to the ORDER clause in ADDL syntax, which specifies whether the order of the current index segment is ascending or descending. For the GDB API, an index segment is assumed to be ascending unless you explicitly set this attribute.

## Data Type Attributes (UDT\_DATATYPE)

#### UDT\_M\_DATE\_

The data type stores date information.

**Notes**

- This setting affects the types of data manipulation that Attunity Connect allows for fields based on this data type. Set the UDT\_M\_DATE\_ attribute to enable operations such as date arithmetic.

**UDT\_M\_NUMBER\_**

The data type stores numeric information.

**Notes**

- Attunity Connect uses this attribute to quickly check data type matching in the SQL statements. If the types of one or more expressions in an SQL statement do not match, Attunity Connect knows that some kind of conversion may be necessary.

**UDT\_M\_SCALED\_**

The data type includes a scale value.

**Notes**

- Attunity Connect uses this attribute to perform basic type-checking in the SQL statements. If the custom data type allows scale, Attunity Connect checks for a valid scale value as part of a field definition using this data type.

**UDT\_M\_NULLABLE\_**

For integer data types (such as int, uint, quad), indicates whether the data type allows a null value.

**Notes**

- Attunity Connect uses this attribute to validate data, both on input and output. If the custom data type does not allow a null value, Attunity Connect checks for a non-null value when performing I/O operations on columns using this data type.

You must set this attribute if the data type definition includes the IS\_NULL\_VAL and SET\_NULL\_VAL functions. For details, see page 142.

**UDT\_M\_SIZED\_**

The data type requires size information in addition to width (length).

**Notes**

- Attunity Connect uses this attribute to determine whether the data type's storage size in the physical buffer is equal to its display width (column\_da->length) or equal to a size calculated from the width.

You must set this attribute if the data type definition includes the WIDTH\_TO\_SIZE function. For details, see page 139.

### **UDT\_M\_SCALED\_SIZE\_**

The size for data type depends on both width (length) and scale.

#### **Notes**

- Attunity Connect uses this attribute to calculate the physical storage size of a data type. Unlike the UDT\_M\_SIZED\_ attribute which is associated with a function (WIDTH\_TO\_SIZE), the UDT\_M\_SCALED\_SIZE attribute causes Attunity Connect to perform only a simple size calculation based on width and scale.
- You do not have to set the UDT\_M\_SCALED\_ attribute in order to use this attribute; if the data type is not scaled, Attunity Connect assumes a scale of zero (0) in its calculations.

### **UDT\_M\_FIXED\_WIDTH\_**

The data type has a fixed width, regardless of the actual data value.

#### **Notes**

- Attunity Connect uses this attribute to determine whether a data value requires additional processing to accommodate a fixed width. For example, suppose a text-based data type has a fixed width of 30 characters. Attunity Connect pads with blanks any value that is less than 30 characters and truncates any value greater than 30 characters.

## **Return Status Codes**

### **GDB\_STATUS**

The GDB API defines the following return status codes (GDB\_STATUS):

GDB\_OK\_  
GDB\_NOT\_OK\_  
GDB\_NOT\_FOUND\_  
GDB\_END\_OF\_STREAM\_  
GDB\_DUPLICATE\_KEY\_IN\_INDEX\_  
GDB\_RESOURCE\_LOCKED\_  
GDB\_COMMAND\_SYNTAX\_ERROR\_  
GDB\_CONSTRAINT\_FAILED\_  
GDB\_NO\_PRIVILEGE\_

The data driver functions should check for these status values and respond appropriately to any error messages.

All exceptions arising from user-written functions generate an appropriate error message in the log file.

## UDT\_STATUS

The UDT API defines the following return status codes (UDT\_STATUS):

UDT\_OK  
UDT\_NOT\_OK  
UDT\_OVERFLOW  
UDT\_TRUNCATED\_OK

The custom data type functions should check for these status values and respond appropriately to any error messages.

# Index

## Symbols

<config> statement  
  types schema 106

## A

abs\_offset 259, 300  
  in COLUMN\_DA 279  
access  
  for application adapter 27  
actual\_size\_read 67  
ACX  
  connection verbs 113  
ADD 38, 41, 42, 257, 294  
  creating metadata 295  
  extended metadata 258, 300  
  supported data types 292  
  table name 272  
add 145  
ADD directory  
  specifying in connect string 295  
ADD\_ROW function 56, 261  
  for a table 275  
ADDL\_TO\_GDBH option 170  
  See also NAV\_UTIL  
ADDON option 21, 25, 133  
addon.def 20, 25, 83, 132, 265, 269  
  application adapter example 21  
  format 25, 132  
  identifying LOAD function 34  
  merging define files 21  
  parsing entries 177  
  registering a Data Connector 25  
  registering a data type 133  
  registering a startup function 151  
  syntax 20, 151  
application adapter  
  defining 24  
  defning 20  
  nav\_util protogen 24  
  sample addon.def entry 21  
  setting up access 27  
application adapter API  
  header files 19

ARR\_ADD 160  
ARR\_BINSET 163  
ARR\_BSEARCH 164  
ARR\_COPY 161  
ARR\_DELETE 159  
ARR\_FREE 162  
ARR\_INSERT 159  
ARR\_NEW 158  
ARR\_RESET 158  
ARR\_SET\_ENTRY 161  
ARR\_SIZE 159  
arrays 259  
  bookmark\_size 273  
  counter column 270, 278  
  dimension 277  
  extended metadata 300  
  number of occurrences 277  
  required function 53  
asterisk 40, 173  
atomic data type 138, 276, 291  
atomic datatype  
  MAX\_DISPLAY\_LENGTH function 294  
attributes 294  
  column 278  
  data type 292  
  driver 257  
  index 281  
  index segment 283  
  table 274  
Attunity Connect Data Dictionary 38, 257  
  See ADD  
Attunity Connect Internal Structures 283  
AUTOINCREMENT clause 279, 301  
aux\_column  
  in COLUMN\_DA 278  
  See also arrays, VARIANT  
auxiliary functions  
  See predefined functions  
AV\_UTIL PROTOGEN  
  types schema 106

## B

base\_dsc 135, 136

base\_id 136  
    in UDT\_DATATYPE 292

binding configuration 25  
    adapter configuration 28

binding file 36, 84  
    connection information 27  
    data connector definition 27  
    properties for a data source 182, 184  
    specifying extended metadata 300

BLOB clause 279, 301

blob stream  
    closing 68  
    opening 65  
    reading 66  
    setting current position 68  
    writing 67

blob\_buffer 67

BLOB\_CLOSE function 68, 262

blob\_handle 67, 68  
    contents 66

BLOB\_OPEN function 65, 262

BLOB\_READ function 66, 262

BLOB\_SEEK function 68

BLOB\_WRITE function 67, 262

bm 52, 53, 54, 56, 57, 58, 59, 66  
    See also bookmarks

bookmark\_size  
    driver 259  
    table 54, 273

bookmarks 66, 296  
    converting 53  
    macros 52, 54, 274  
    required function 52  
    required usage 259  
    string representation 53

buffer 45  
    columns contained in 295

buffer column  
    highest value 144  
    lowest value 144  
    random value 145

buffer\_data  
    in GDB\_BUFFER 270

buffer\_length  
    in GDB\_BUFFER 270

buffer\_size 67

**C**

cache  
    use in query processing 297

cardinality 43, 303  
    in INDEX\_DA 281  
    in SEG\_DA 283  
    in TABLE\_DA 274

CASE  
    specifying metadata for 276

case sensitive 258, 297

case\_value  
    in COLUMN\_DA 278  
    See also aux\_column

chapter 279, 302

CHAPTER clause 280

chapter column 280

chapter identifier 280

CHAPTER OF clause 280

chapter stream  
    opening 74  
    setting parent row 75

chapter\_io\_stream 74, 75

chapter\_of  
    in COLUMN\_DA 280

chapter\_table\_da 75  
    in COLUMN\_DA 280

chapter\_value 75

chapters 73, 74

child table  
    See chapter

CLOSE function 33, 54, 260  
    for a table 275

CLUSTERED clause 282, 303

Codepage Support Functions 176

column 284  
    attributes 301

column lists 46, 270, 295  
    insert operations 295  
    n\_columns 270  
    See also  
        GDB\_M\_SUPPORT\_COLUMN\_LIST  
        ST\_

column\_da  
    in GDB\_BUFFER\_COLUMN 271

COLUMN\_DA structure 139, 275  
    attributes 301  
    specifying a table or procedure 273

specifying parameters for procedure 273  
**column\_name**  
 in COLUMN\_DA 276  
**column\_number** 66  
 embedded chapter 74  
 in GDB\_BUFFER\_COLUMN 271  
 in GDB\_FTREE\_NODE 284  
 in SEG\_DA 283  
**columns**  
 in GDB\_BUFFER 270  
 in TABLE\_DA 273  
**command** 274, 301  
 cursor handle 69  
 executing 70  
 executing with parameters 71  
 freeing resources 72  
 reusing 71  
**command\_text** 69, 71  
 in TABLE\_DA 273  
**commands**  
 parameters 51  
 See SQL commands  
**COMMIT\_TRANSACTION** function 60, 261  
**COMPARE** 162  
**compare**  
 in UDT\_DATATYPE 293  
**COMPARE** function 140, 293  
**comparison**  
 native 141  
**config statement** 28  
 types schema 106  
**CONNECT** function 33, 35, 259  
**connect string**  
 for an ADD driver 295  
**connect\_string** 36  
**connecting**  
 via ACX 113  
**Connecting to data**  
 GDB API 35  
**connection\_information**  
 binding file 27  
**const\_string**  
 in GDB\_FTREE\_NODE 284  
**constant** 284  
**constants** 50  
**control fields** 276  
 See also STRUCTURE, VARIANT, CASE

CONVERT\_BM\_DATA function 53, 261  
**convert\_to** 293  
**CONVERT\_TO** function 136, 293  
 supplying data type ids 294  
**convert\_to\_datatypes** 136, 294  
**counter column** 278  
**CP\_GET\_ID** 177  
**CP\_INFO** 175  
**CP LOWERCASE** 176  
**CP REGISTER** 177  
**CP REGISTER\_CVT** 177  
**CP STRCHR** 175  
**CP STRICMP** 175  
**CP STRNICMP** 176  
**CP STRNW CMP** 176  
**CP STRSTR** 175  
**CP TOUPPER** 176  
**CP UPPERCASE** 175  
**CREATE INDEX** statement 63  
**CREATE TABLE** statement 62  
**CREATE\_INDEX** function 63, 262  
**CREATE\_TABLE** function 62, 262  
 valid data types 62  
**custom adapter**  
 defining 24  
 setting up access 27  
**custom driver**  
 access to data sources 25  
 attributes 294  
 defining 23  
 how to register 25  
 sample CLOSE 197  
 sample CONNECT 194  
 sample DISCONNECT 195  
 sample FETCH 197  
 sample functions 193  
 sample GET\_LAST\_ERROR 198  
 sample LOAD 198  
 sample OPEN 195  
 sample REWIND 198  
 sample SET\_BUFFER 196  
 sample SET\_INDEX 196  
 setting up access to data 26  
 setup 193

**D**

data

in GDB\_VAL\_DESC 267

data access

- for data connector 26

data buffer structures 266

Data Connector

- metadata 38

data connector

- define file format 25, 132
- function flow 28
- setting up access to data 26
- troubleshooting 28

Data driver

- name 20

data driver

- defining 23

data source 38

- setting properties 182, 184

data source type

- See also ODBC driver name

data structures 255

data type

- assignment routines 143, 144, 145
- comparison routines 140, 141, 142
- conversion routines 134, 135, 136, 137, 139
- defining 131
- exposed 141
- exposed size 137
- for data type registration 132
- name 20, 132
- registering on server 132

See also addon.def

- structure 290

support functions 154

- support macros 147

data types

- header files 19
- in CREATE\_TABLE 62

data\_connector\_name

- binding file 27

data\_offset

- in GDB\_BUFFER\_COLUMN 271

Database Property Functions 183

datatype

- in COLUMN\_DA 276

date

- converting to literal 72

date field

- user-defined 292, 303

date\_literal 73

date\_value 73

db\_command

- in COLUMN\_DA 279
- in INDEX\_DA 282
- in SEG\_DA 283
- in TABLE\_DA 274

DB\_COMMAND clause 183, 274, 279, 282, 283

DB\_GET\_PROPERTY 183

DB\_GET\_PROPERTY\_LONG 184

DB\_GET\_PROPERTY\_STRING 183

DB\_SET\_PROPERTY 184

dbgdb.h 131, 132, 255

- GDB API 19
- sample use 193, 247

decimal data type 277

DEF directory 20

default\_format 293

DEFAULT\_FORMAT function 293

define file 21

- for add-on definitions 132

DEFINE PROCEDURE statement

- table name 272

DEFINE RECORD statement

- table name 272

define\_file 21

DELETE\_ROW function 57, 261

- for a table 275

DESCRIBE\_CHAPTER function 73, 263

DESCRIBE\_COMMAND function 69, 70, 262, 274

DESCRIBE\_SP function 41, 260, 268, 272

DESCRIBE\_TABLE function 40, 259, 268, 272

description

- in COLUMN\_DA 278
- in GDB\_ITEM\_DESC 268
- in INDEX\_DA 281
- in TABLE\_DA 274
- in UDT\_DATATYPE 291

DESCRIPTION clause 274, 278, 281, 291

Developer SDK

- data structures 255
- definition file 177
- header files 19
- installation 19
- predefined functions 153

setup tasks 19  
**dimension\_1** 53  
 in COLUMN\_DA 277  
**dimension\_2**  
 in COLUMN\_DA 277  
**DISCONNECT function** 38, 259  
**display width** 139  
**DLL**  
 exposing INIT-FUNCTION 26, 84, 131, 133,  
     152  
 exposing LOAD function 34  
 for custom ODBC driver 83  
 for GDB and UDT functions 25  
 for ODBC functions 84  
 for startup function 151  
 for startup functions 132, 151  
 locating an ODBC driver's INIT-FUNC-  
     TION 84  
**driver\_type**  
 in GDB\_FILE\_INFO 269  
**Drivers**  
 custom 20, 25  
 See also custom driver, GDB API  
**DROP TABLE statement** 64  
**DROP\_INDEX function** 65, 262  
**DROP\_TABLE function** 64, 262  
**ds\_name** 36  
**DSTR\_APPEND** 165  
**DSTR\_CUT** 167  
**DSTR\_GET\_ITEM** 168  
**DSTR\_INSERT** 167  
**DSTR\_LENGTH** 166  
**DSTR\_M\_APPEND** 166  
**DSTR\_NEW** 165  
**DSTR\_RESET** 166  
**DT\_ADD\_DATATYPE** 132, 147, 154  
**DT\_CONVERT** 147, 154  
**dtypeid.h** 131, 132, 267, 276, 291, 292  
 data types 19  
 sample use 247  
**dynamic array functions** 157  
**dynamic string functions** 164

**E**

**end\_buffer** 48  
**environment variable**  
 use in Developer SDK 253

**error code** 76, 146  
**Error function**  
 in GDB API 76  
 in UDT API 146  
**error messages** 305, 306  
 See GDB\_STATUS  
 See UDT\_STATUS  
**error text** 76  
**EXECUTE\_IMMEDIATE function** 70, 262  
**EXECUTE\_WITH\_PARAMS function** 71, 263  
**EXPLICIT\_SELECT clause** 279, 301  
**exposed\_desc** 138  
**extended metadata** 281, 300

**F**

**FETCH function** 33, 53, 75, 260  
 bookmarks 54  
 for a table 275  
 processing unique index 303  
**field**  
 allowing null values 278, 279, 280, 301, 302  
**file** 43, 178  
 See also Profile Functions  
**file\_organization**  
 in GDB\_FILE\_INFO 269  
**FILENAME clause** 273  
 See also GDB\_FILE\_INFO  
**filter expressions** 46, 48, 258, 298  
 constants 50  
 for multiple index segments 299  
 GDB-related macros 298, 299  
 including index columns 299  
 macros 48, 50  
 order of execution 50  
 parameter usage 298  
 parsing 49  
 passing parameters 258  
 setting parameters 50  
**Filter Function** 170  
**filter structure**  
 See GDB\_FTREE\_NODE  
**filter tree** 47  
**filter\_consts** 49  
 example 286  
**filter\_param**  
 example 286  
**filter\_params** 51

filter\_tree 49  
    example 286  
filters  
    tree example 285  
fixed width datatype  
    MAX\_DISPLAY\_LENGTH 294  
FRACTIONS clause 277  
FREE\_COMMAND function 72, 263  
from\_base 293  
FROM\_BASE function 135, 293  
from\_string 293  
FROM\_STRING function 134, 293  
    exposed data type 138  
function  
    See INIT-FUNCTION  
    Tandem restrictions 254  
function flow  
    data connector 28  
function structures 255  
functions  
    See also startup function  
    startup 20, 151

## G

gap.h  
    application adapter API 19  
GDB API  
    blob support 65  
    chapter support 73  
    command support 68  
    connecting to a data source 35  
    data definition 61, 296  
    defining metadata 38  
    header files 19  
    loading a driver 34  
    macros 41, 42, 48, 50, 52, 54, 56, 57, 58, 59,  
        60, 257, 268, 274, 291, 298, 299  
    minimum required functions 33  
    required functions 33, 34, 35, 44, 45, 54, 55  
    retrieving data 44  
    tracing 265  
    transactions 59  
    types of functions 33, 112  
    updating data 56  
    version 265  
GDB\_BUFFER structure 46, 48, 51, 54, 57, 270  
GDB\_BUFFER\_COLUMN structure 270

GDB\_COL\_M\_AUTOINCREMENT\_ 279, 301  
GDB\_COL\_M\_BLOB\_ 65, 66, 67, 68, 279, 301  
GDB\_COL\_M\_CHAPTER\_ 279, 302  
GDB\_COL\_M\_EXPLICIT\_SELECT\_ 279, 301  
GDB\_COL\_M\_HIDDEN\_ 279, 302  
GDB\_COL\_M\_NON\_UPDATEABLE\_ 279, 301  
GDB\_COL\_M\_NULL\_FLD\_DESC\_ 279, 280,  
    302  
GDB\_COL\_M\_NULLABLE\_ 278, 301  
gdb\_command 69, 70, 72  
    in TABLE\_DA 274  
GDB\_DB\_FUNCTIONS 34, 199, 255  
    attributes 294  
    for a table 274  
    io\_add\_row 56  
    io\_blob\_close 68  
    io\_blob\_open 65  
    io\_blob\_read 66  
    io\_blob\_seek 68  
    io\_blob\_write 67  
    io\_close 54  
    io\_commit\_transaction 60  
    io\_connect 35  
    io\_convert\_bm\_data 53  
    io\_create\_index 63  
    io\_create\_table 62  
    io\_delete\_row 57  
    io\_describe\_chapter 73  
    io\_describe\_command 69  
    io\_describe\_sp 41  
    io\_describe\_table 40  
    io\_disconnect 38  
    io\_drop\_index 65  
    io\_drop\_table 64  
    io\_execute\_immediate 70  
    io\_execute\_with\_params 71  
    io\_fetch 53  
    io\_free\_command 72  
    io\_get\_date\_literal 72  
    io\_get\_extended\_metadata 42  
    io\_get\_file\_info 43  
    io\_get\_item\_list 39  
    io\_get\_last\_error 76  
    io\_lock\_row 58  
    io\_open 44  
    io\_open\_chapter 74  
    io\_prepare\_command 69

---

io\_prepare\_commit\_transaction 61  
 io\_recover\_transaction 61  
 io\_rewind 55  
 io\_rollback\_transaction 60  
 io\_set\_active\_chapter 75  
 io\_set\_bm 51  
 io\_set\_buffer 45  
 io\_set\_connect 36  
 io\_set\_filter 48  
 io\_set\_filter\_params 50  
 io\_set\_index 46  
 io\_set\_params 51  
 io\_start\_distributed\_transaction 61  
 io\_start\_transaction 59  
 io\_unload 35  
 io\_unlock\_row 58  
 io\_update\_row 56  
 sql\_syntax\_name 300  
**gdb\_db\_functions**  
 See GDB\_DB\_FUNCTIONS  
**GDB\_DEBUG\_MODE** 265  
**gdb\_debug\_mode** 265  
 See also Troubleshooting  
**GDB\_EQUAL\_**  
 See index\_relation  
**GDB\_FILE\_INFO** structure 44, 268  
**GDB\_FTREE\_NODE** structure 284, 298  
**GDB\_FTREE\_OPERATOR** enumeration 284  
**GDB\_GET\_TABLE\_DA** 173  
**GDB\_GREATER\_EQUAL\_**  
 See index\_relation  
**gdb\_handle** 60, 62, 64, 66, 69, 71, 73, 74, 76  
 in CONNECT 36, 37  
 in DESCRIBE\_SP 42  
 in DESCRIBE\_TABLE 41  
 in DISCONNECT 38  
 in GET\_EXTENDED\_METADATA 43  
 in GET\_ITEM\_LIST 40  
 in OPEN 45  
**gdb\_help**  
 See GDB\_HELP\_FUNCTIONS structure  
**GDB\_HELP\_DEFINE**  
 sample use 194, 247  
**GDB\_HELP\_DEFINE** macro 132, 153  
**GDB\_HELP\_FUNCTIONS** structure 34, 132, 153, 194, 199, 255, 263  
 initializing function pointer 131  
**GDB\_HELP\_USE** macro 153  
**GDB\_HELP\_VERSION\_NUMBER** 265  
**GDB\_INDEX\_M\_BEST\_UNIQUE** 282, 303  
**GDB\_INDEX\_M\_CLUSTERED** 282, 303  
**GDB\_INDEX\_M\_HASHED** 282, 302  
**GDB\_INDEX\_M\_HIERARCHY** 282, 303  
**GDB\_INDEX\_M\_UNIQUE** 282, 302  
**GDB\_INITIALIZE** macro 131, 153, 199, 257, 291  
**GDB\_ITEM\_DESC** structure 40, 267  
**GDB\_ITEM\_TYPE** 39, 268  
 macros 268  
**GDB\_M\_ADD** 257, 294  
**GDB\_M\_CASE\_SENSITIVE** 258, 272, 297  
**GDB\_M\_DISABLE\_GST** 258, 297  
 effect on multi-threading 297  
**GDB\_M\_FILTER\_WO\_INDEX** 47, 48, 258, 299  
**GDB\_M\_FILTER\_WO\_SEGMENTS** 47, 48, 258, 299  
**GDB\_M\_INSERT\_ALL\_COLUMNS** 258, 295  
**GDB\_M\_SUPPORT\_ABS\_OFFSET** 259, 279, 300  
**GDB\_M\_SUPPORT\_COLUMN\_LIST** 258, 295  
 See also column lists  
**GDB\_M\_SUPPORT\_EXTENDED\_NAMES** 258, 300  
**GDB\_M\_SUPPORT\_FILTER\_PARAMS** 258, 298  
**GDB\_M\_SUPPORT\_FILTERS** 258, 298, 299, 300  
**GDB\_M\_SUPPORT\_MULTI\_THREAD** 258, 296, 298  
**GDB\_M\_SUPPORT\_PASSTHRU\_COMMAND** 69, 70, 258, 297  
**GDB\_M\_SUPPORT\_SQL** 69, 70, 259, 300  
**GDB\_M\_SUPPORT\_TRANS** 258, 296  
**GDB\_M\_SUPPORT\_UPDATE** 258, 296  
**GDB\_M\_TABLE\_COMMAND** 70  
**GDB\_PROF\_DELETE** 182  
**GDB\_PROF\_GET\_CUR\_VALUE** 181  
**GDB\_PROF\_GET\_VALUE** 180  
**GDB\_PROF\_INIT\_FILE** 178  
**GDB\_PROF\_ITEM\_FIRST** 179  
**GDB\_PROF\_ITEM\_NEXT** 180  
**GDB\_PROF\_SECTION\_FIRST** 178  
**GDB\_PROF\_SECTION\_NEXT** 179  
**GDB\_PROF\_SET\_ITEM** 181  
**GDB\_PROF\_SET\_SECTION** 181

GDB\_SEG\_M\_DESCENDING\_ 283, 303  
GDB\_STATUS 305  
GDB\_TABLE\_M\_COMMAND\_ 273, 274, 301  
GDB\_TABLE\_M\_CONVERT\_BUF\_ 301  
GDB\_TABLE\_M\_STORED\_PROC\_ 274, 301  
GDB\_TRAVERSE\_FILTER function 298  
GDB\_TREENODE enumeration 284  
GDB\_VAL\_DESC structure 49, 51, 72, 73, 134, 135, 137, 138, 141, 142, 143, 144, 145, 266, 271  
    for chapters 75  
GDB\_VERSION\_NUMBER\_ 257, 291  
GDBH\_DESCRIBE\_SP 171  
GDBH\_DESCRIBE\_TABLE 171  
GDBH\_GET\_ITEM\_LIST 172  
gdbTrace 28  
General Service Thread (GST) 298  
general string functions 168  
GET\_DATE\_LITERAL function 72, 263  
GET\_EXTENDED\_METADATA function 42, 260  
GET\_FILE\_INFO function 43, 260  
GET\_ITEM\_LIST function 39, 41, 42, 259, 267  
get\_last\_error  
    in UDT\_DATATYPE 293  
GET\_LAST\_ERROR function 259, 293  
    in GDB API 76  
    in UDT API 146

## H

HASHED clause 282, 302  
header files 19, 132, 255, 267, 276  
HIDDEN clause 279, 302  
hidden columns  
    See GDB\_COL\_M\_HIDDEN\_  
hierarchical data 303  
highest\_val 293  
HIGHEST\_VAL function 144, 293

## I

ICOLUMN\_DA  
    See Attunity Connect Internal Structures  
id  
    in GDB\_VAL\_DESC 267  
    in UDT\_DATATYPE 291  
    specifying column's data type 276

IINDEX\_DA  
    See Attunity Connect Internal Structures  
image  
    See SHAREABLE-NAME keyword  
    Tandem restrictions 254  
image value  
    OpenVMS 253  
    OS/390 254  
    Tandem 254  
    UNIX 253  
    Windows 95/98/NT 253  
INCLUDE directory 19  
index 46, 258  
    attributes 302  
    cardinality 281  
    hierarchical data 303  
index relation 55  
index segment  
    attributes 303  
    number of 281  
    processing filter expressions 299  
INDEX\_DA structure 280  
    attributes 302  
    specifying a table or procedure 273  
index\_id  
    in INDEX\_DA 281  
INDEX\_ID clause 281  
index\_length  
    in INDEX\_DA 281  
index\_name  
    in INDEX\_DA 280  
index\_number 47, 64  
index\_relation 47  
indexs  
    in TABLE\_DA 273  
indicator  
    in GDB\_VAL\_DESC 267  
indicator\_offset  
    in GDB\_BUFFER\_COLUMN 271  
INIT-FUNCTION 34  
    exposing 26, 84, 133, 152  
    for startup function 151  
INIT-FUNCTION keyword 25, 132, 151, 265  
    for driver LOAD function 34  
    for ODBC driver 84  
INSERT command  
    restrictions 279

---

insert operations 295  
**internal\_use**  
     in COLUMN\_DA 280  
     in INDEX\_DA 282, 283  
     in SEG\_DA 283  
     in TABLE\_DA 275  
**io\_add\_row** 261  
**io\_blob\_close** 262  
**io\_blob\_open** 262  
**io\_blob\_read** 262  
**io\_blob\_seek** 262  
**io\_blob\_write** 262  
**io\_close** 260  
**io\_commit\_transaction** 261  
**io\_connect** 259  
**io\_convert\_bm\_data** 261  
**io\_create\_index** 262  
**io\_create\_table** 262  
**io\_delete\_row** 261  
**io\_describe\_chapter** 263  
**io\_describe\_command** 262  
**io\_describe\_sp** 260  
**io\_describe\_table** 259  
**io\_disconnect** 259  
**io\_drop\_index** 262  
**io\_drop\_table** 262  
**io\_execute\_immediate** 262  
**io\_execute\_with\_params** 263  
**io\_fetch** 260  
**io\_free\_command** 263  
**io\_get\_date\_literal** 263  
**io\_get\_extended\_metadata** 260  
**io\_get\_file\_info** 260  
**io\_get\_item\_list** 259  
**io\_get\_last\_error** 259  
**io\_lock\_row** 261  
**io\_open** 260  
**io\_open\_chapter** 263  
**io\_prepare\_command** 262  
**io\_prepare\_commit\_transaction** 261  
**io\_recover\_transaction** 262  
**io\_rewind** 260  
**io\_rollback\_transaction** 261  
**io\_set\_active\_chapter** 263  
**io\_set\_bm** 260  
**io\_set\_buffer** 260  
**io\_set\_connect** 259  
           **io\_set\_filter** 260  
           **io\_set\_filter\_params** 260  
           **io\_set\_index** 260  
           **io\_set\_params** 260  
           **io\_start\_distributed\_transaction** 261  
           **io\_start\_transaction** 261  
           **io\_stream** 45, 46, 47, 49, 51, 52, 53, 54, 55, 56, 57,  
               58, 59  
           **io\_unload** 259  
           **io\_unlock\_row** 261  
           **io\_update\_row** 261  
           **is\_empty\_val** 293  
           **IS\_EMPTY\_VAL function** 141, 143, 293  
           **is\_null\_val** 293  
           **IS\_NULL\_VAL function** 142, 293  
               See also UDT\_M\_NULLABLE\_  
**ISEG\_DA**  
     See Attunity Connect Internal Structures  
**isolation level** 59  
**ITABLE\_DA**  
     See Attunity Connect Internal Structures  
**item** 179, 180, 182  
     See also Profile Functions  
**item\_desc** 40  
**item\_type** 40, 172, 173  
     See also GDB\_ITEM\_TYPE

## K

**keys** 269  
     See also indices

## L

**length** 139, 292, 304, 305  
     in COLUMN\_DA 276  
**index** 281  
     of allocated buffer 270  
     See also precision  
     See also size  
**LIBNAVA library**  
     use with Developer SDK 254  
**LIKE operator**  
     **GDB\_FTREE\_OPERATOR enumeration**  
         285  
**list tables**  
     **GET\_ITEM\_LIST function** 39  
**LOAD function** 33, 34

LOCAL\_COPY 41, 42, 45  
lock\_mode 52  
LOCK\_ROW function 58, 261  
    for a table 275  
log file 28  
    Developer SDK error messages 26, 84, 133, 151  
logical name  
    use in Developer SDK 253  
lowest\_val 293  
LOWEST\_VAL function 144, 293

## M

macros  
    ADD 41, 42  
    bookmarks 52, 54, 274  
    driver 298, 299  
    filter expressions 298, 299  
    filters 48, 50  
    for item type 268  
    GDB\_INITIALIZE 199, 257, 291  
    transaction 59, 60  
    UDT\_ADD\_DATATYPE 265, 290  
    UDT\_CONVERT 265  
        update 56, 57, 58  
mask 40  
max\_display\_length 139, 294  
MBLEN 174  
memory pool functions 155  
MEMP\_DELETE 157  
MEMP\_FREE 156  
MEMP\_FREE\_POOL 157  
MEMP\_MALLOC 156  
MEMP\_MALLOC8 156  
MEMP\_NEW 155  
metadata 33, 38, 274  
    cached 62, 63  
    chapter 73, 74  
    extended 53  
    parent table 74  
    structures 271  
    support functions 170  
mode 45  
    for bookmark conversion 53  
multi-threading 258, 296, 297

## N

n\_affected\_rows 71, 72  
n\_columns  
    in GDB\_BUFFER 270  
    in TABLE\_DA 273  
n\_filter\_consts 49  
    example 286  
n\_filter\_tree 49  
    example 285  
n\_indexs  
    in TABLE\_DA 273  
n\_keys  
    in GDB\_FILE\_INFO 269  
n\_params 51, 72  
    example 286  
    in TABLE\_DA 273  
n\_rows  
    in GDB\_FILE\_INFO 269  
N\_ROWS clause 269  
    index 281  
    table 274  
n\_segments  
    in INDEX\_DA 281  
name  
    for custom driver 269  
    for user-defined data type 291  
    in GDB\_ITEM\_DESC 268  
    in startup function definition 151  
    setting in addon.def 20, 25, 84, 132, 151  
nav.log 28, 76, 146  
nav\_register\_function  
    fnc\_class 254  
    fnc\_name 254  
    use on Tandem platforms 254  
NAV\_UTIL  
    ADDL\_TO\_GDBH 170  
    ADDON option 21, 25, 133, 151  
    enabling Developer SDK 254  
    LOCAL\_COPY 41, 42  
    update\_statistics 281  
nav\_util protogen 24  
nest\_level  
    in COLUMN\_DA 277  
    See also STRUCTURE, VARIANT  
NLS functions 174  
NLS\_CVTFNC 174  
NON\_UPDATEABLE clause 279, 301

non-atomic data type 138  
 non-atomic datatype  
   WIDTH\_TO\_SIZE function 139  
 null indicator 267  
   offset 271  
   See also indicator  
 null value 278, 279, 280, 301, 302  
   user-defined 292, 304  
 null\_desc  
   in COLUMN\_DA 280  
 NULL\_VALUE clause 279, 302  
 Nullable clause 278, 301  
 num 145  
 numeric field  
   user-defined 292, 304

**O**

ODBC  
   APIs 83, 265  
   custom driver 83  
   registering custom driver 83  
 ODBC API  
   header files 19  
 ODBC driver  
   access to data sources 83  
   define file syntax 83  
   defining 20  
   name 20, 84  
   See also addon.def  
 ODBC\_FUNCTIONS structure 84, 255, 265  
 odbcfn.h 83, 255  
   ODBC API 19  
 OFFSET IS clause 279  
 OPEN function 33, 44, 260  
   for a table 274  
 OPEN\_CHAPTER function 74, 263  
 OpenVMS  
   image value 253  
 operator 284  
 ORDER clause 283, 303  
 ORGANIZATION clause 269  
 OS/390  
   image value 254  
 owner support 37

**P**

parallel execution 258  
 param\_number  
   in GDB\_FTREE\_NODE 284  
   See also params  
 parameters 50, 258, 271, 284, 298  
   for commands 51  
   for stored procedure 51  
   input to command 71  
 params 51, 72  
   in TABLE\_DA 273  
   See also parameters  
 parent\_chapter\_column 75  
 parent\_io\_stream 74  
 parent\_table\_da 74  
 PASSTHRU command 258, 297, 300  
 path  
   in TABLE\_DA 273  
 physical\_file\_name  
   in GDB\_FILE\_INFO 269  
 platform  
   Developer SDK restrictions 253  
 precision 292  
   in COLUMN\_DA 277  
   See also length  
   See width  
 predefined functions 153  
   in GDB\_HELP\_FUNCTIONS structure  
   use with custom data types 132  
 PREPARE\_COMMAND function 69, 70, 262, 274  
 PREPARE\_COMMIT\_TRANSACTION function  
   61, 261  
 prof 178, 179, 180, 181, 182  
   See also Profile Functions  
 property 183, 184, 185  
   modifying 182  
   See also Database Property Functions

**Q**

query optimization 46, 302, 303  
 Query Processor  
   index cardinality 281  
   table cardinality 274  
 question mark 40, 173

**R**

random\_val 293  
RANDOM\_VAL function 145, 293  
range 48

    end value 48  
    minimum values 48  
    setting index values 48

read-write I/O processing  
    See update operations

read-write operations 259, 273

record size 43

record\_format  
    in GDB\_FILE\_INFO 269

RECORD\_FORMAT clause 269

record\_size

    in GDB\_FILE\_INFO 269

RECOVER\_TRANSACTION function 61, 262

registration function 131, 291

    See also INIT-FUNCTION

request 40

requested\_size 67

res\_dsc 137

reserved keywords 272, 276, 280

RESET 174

resultset 70

return codes

    GDB API 305

    UDT API 305

REWIND function 33, 55, 260

    for a table 275

ROLLBACK\_TRANSACTION function 60, 261

root node

    See filter expressions

row

    deleting 57

    inserting 56

    locking 58

    unlocking 58

    updating 57

**S**

scale 305

    in COLUMN\_DA 277

    in GDB\_VAL\_DESC 267

    user-defined 292

SCALE clause 277

scale value

    user-defined 304

scaled data type 277

search conditions 48

section 179, 180, 181

    See also Profile Functions

security

    hook 77

    monitoring security hooks 78

    setting up security hooks 78

See also Codepage Support Functions

See also Database Property Functions

See also dynamic array functions

See also dynamic string functions

See also general string functions

See also memory pool functions

See also metadata support functions

See also NLS functions

See also Profile Functions

SEG\_DA structure 281, 282

    attributes 303

segments

    in INDEX\_DA 281

sequential read 46

SET\_ACTIVE\_CHAPTER function 75, 263

SET\_BM function 51, 260

    for a table 275

SET\_BUFFER

    for current stream 54, 56

SET\_BUFFER function 33, 45, 260

    array processing 270

    for a table 274

SET\_CONNECT function 36, 259

set\_empty\_val 293

SET\_EMPTY\_VAL function 141, 143, 293

SET\_FILTER function 48, 260

    behavior with SET\_INDEX 46

    filter tree example 285

    for a table 274

    use with index columns 299

SET\_FILTER\_PARAMS function 50, 260, 284,

    298

    filter tree example 286

    for a table 274

SET\_FIXED\_WIDTH function 294

SET\_INDEX function 46, 50, 260

    for a table 274

    overriding default behavior 299

**set\_null\_val** 293  
**SET\_NULL\_VAL** function 142, 143, 293  
     See also **UDT\_M\_NULLABLE**  
**SET\_PARAMS** function 51, 260  
     for a table 274  
**shareable image**  
     for GDB and UDT functions 25  
     for ODBC functions 84  
     for startup functions 132, 151  
     See also **DLL**  
     See also **DLL, SHAREABLE-NAME keyword**  
**SHAREABLE-NAME**  
     OpenVMS 253  
     OS/390 254  
     platform restrictions 253  
     Tandem 254  
     UNIX 253  
     Windows 95/98/NT 253  
**SHAREABLE-NAME** keyword  
     for ODBC driver 84  
     for startup function 151  
     use in **addon.def** 25, 132  
**shared library**  
     location 253  
     See also **DLL**  
**size** 304, 305  
     for user-defined data type 292  
     in **GDB\_VAL\_DESC** 267  
     in **UDT\_DATATYPE** 291  
**SIZE** clause 269, 276, 277  
**sp\_name** 42  
**SQL** command 259, 300  
**SQL** commands 259, 271  
**sql\_syntax\_name** 259  
**src\_desc** 138  
**start\_buffer** 48  
**START\_DISTRIBUTED\_TRANSACTION** function 61, 261  
**START\_TRANSACTION** function 59, 261  
**startup function**  
     for data type registration 132  
     name 20, 151  
     registering 151  
     sample **addon.def** entry 20  
     usage 151  
     when called 151  
**status** 182, 185  
     for **GDB\_FILE\_INFO** structure 43  
     in **GDB\_FILE\_INFO** 269  
**status codes** 305  
**stored procedure** 42, 274, 301  
     parameters 51  
**stored procedures** 273  
     See also **parameters**  
**STR\_C\_STRING** 168  
**STR\_C\_STRING\_PAD** 169  
**str\_dsc** 134, 135  
**STR\_STRCONCAT** 169  
**stream**  
     bookmarks in 52  
     closing 55  
     opening 44  
     reusing 55  
**STRUCTURE**  
     grouping level 277  
     specifying metadata for 276  
     use with arrays 277  
**support functions**  
     See **predefined functions**  
**syntax file** 259  
**system restrictions**  
     shareable image 253

## T

**table**  
     attributes 301  
     cardinality 274  
     supplying metadata for 40  
**table\_da** 41, 42, 43, 45, 62, 64, 65, 66, 69, 70, 74  
**TABLE\_DA** structure 41, 42, 43, 45, 272  
     attributes 301  
**table\_da\_changed** 43  
**table\_db\_functions**  
     in **TABLE\_DA** 274  
**table\_id**  
     in **TABLE\_DA** 272  
**table\_name** 41  
     in **TABLE\_DA** 272  
**tables**  
     owner support 37  
**Tandem**  
     image value 254  
     registering custom functions 254

**TDP**

See also data source

**TDP\_EXTENDED\_DIR** 300

**tdp\_name** 37, 183, 184, 185

See also Database Property Functions

**TDP\_PROPERTIES** statement 182, 184

**TDP-NAMES** section 84

**to\_base** 293

**TO\_BASE** function 135, 293

**to\_exposed** 294

**TO\_EXPOSED** function 137

**to\_string** 292

**TO\_STRING** function 134, 292

    exposed data type 138

**tracing** 265

    debug mode 265

**tracing operations**

    data connector 28

**GDB\_DEBUG\_MODE** 265

**tran\_mode** 60

**transactions** 45, 258, 296

    committing 60

    delayed locking 58

    rolling back 60

    starting 59

**TRAVERSE\_FILTER** 170

**TRAVERSE\_FILTER** function 50

**treenode\_type** 284

**two-phase commit** 61

**type**

    in **GDB\_ITEM\_DESC** 268

    macros 268

    See **TYPE keyword**

**TYPE keyword**

    for ODBC driver 83

    for startup function 151

    use in **addon.def** 20, 25, 132

    valid settings

**types schema** 105

**config** 106

**NAV\_UTIL PROTOGEN** 106

**U**

**UDT API** 131

    assignment functions 142

    comparison functions 140

    conversion functions 133

    error functions 146

    header files 19

    initialization functions 142

    macros 265, 290

    registering data type on server 131

**UDT\_ADD\_DATATYPE** macro 131, 147, 155, 265, 290

**UDT\_CONVERT**

    sample use 247

**UDT\_CONVERT** macro 147, 154, 265

**UDT\_DATATYPE** structure 131, 133, 138, 142, 144, 255, 267, 290

    attributes 303

    compare 140

    convert\_to 136

    converting data 136, 139

    from\_base 135

    from\_string 134

    get\_last\_error 146

    highest\_val 144

    is\_empty\_val 141

    is\_null\_val 142

    lowest\_val 144

    name 132

    random\_val 145

    set\_empty\_val 143

    set\_null\_val 143

    to\_base 135

    to\_exposed 137

    to\_string 134

    width\_to\_size 139

**udt\_dsc** 134, 135, 136, 137, 142, 143, 144, 145

**UDT\_END\_CONVERT\_LIST** 294

**UDT\_M\_DATE\_** 73, 292, 303

**UDT\_M\_FIXED\_WIDTH\_** 292, 305

**UDT\_M\_NULLABLE\_** 142, 144, 292, 304

**UDT\_M\_NUMBER\_** 292, 304

**UDT\_M\_SCALED\_** 292, 304, 305

**UDT\_M\_SCALED\_SIZE\_** 292, 305

**UDT\_M\_SIZED\_** 139, 292, 304

**UDT\_STATUS** 306

**udt1\_dsc** 141

**udt2\_dsc** 141

**UNIQUE** clause 282, 302

    See also **GDB\_INDEX\_M\_BEST\_UNIQUE\_**

**UNIX**

    image value 253

- 
- UNLOAD function 35, 259  
**UNLOCK\_ROW** function 58, 261  
     for a table 275  
**update** operations 56, 67, 258, 259, 296  
     buffer contents 295  
     for a column list 295  
     See also Updating Data  
**UPDATE\_ROW** function 56, 261  
     for a table 275  
**update\_statistics**  
     See NAV\_UTIL 281  
**user-defined data type**  
     name 20  
**user-defined data types**  
     attributes 303  
     sample 247  
     See also data type  
**username\_password** 36
- V**
- val** 142, 143, 144  
**val\_desc**  
     in GDB\_BUFFER\_COLUMN 271  
**value** 180, 181, 183, 184, 185  
     See also Profile Functions  
**value\_type** 185  
**VARIANT**  
     discriminating field 278  
     grouping level 277  
     specifying metadata for 276  
**VARIANT VALUE** clause 278  
**verbs**  
     connection 113  
**version** 131, 199  
     for predefined functions 264  
     GDB\_HELP\_VERSION\_NUMBER\_ 265  
     in GDB API 257  
     in UDT API 291  
**virtual table** 53  
**virtual tables** 300  
     See also extended metadata
- W**
- WHERE** clause 48  
**width** 139, 304, 305  
     fixed 292  
     for user-defined data type 292  
     in GDB\_VAL\_DESC 267  
**width\_to\_size** 294  
**WIDTH\_TO\_SIZE** function 138, 139, 294  
     See also UDT\_M\_SIZED\_  
     wildcards 40, 173  
**Windows 95/98/NT**  
     image value 253
- X**
- xmlproto.h**  
     application adapter API 19

